uc3m | Universidad Carlos III de Madrid

Bachelor's Degree in Energy Engineering
2016/2017

Bachelor's Thesis

Simulation of Quantum Algorithms

Guillermo García Matey

Fernando Barbero González (CSIC)
Eduardo Jesús Sánchez Villaseñor (UC3M)
Madrid 16^{th} of October of 2017

Contents

List of Figures

1 Introduction

In this TFG I am going to discuss the basics of quantum algorithms. I will focus on two of the most important ones: the Grover and the Shor algorithms. I will start by giving a brief introduction to quantum physics with the purpose of describing the main tools used in quantum computation and, also, to introduce conventions and notation. In this part I will follow standard references such as [1, 2].

After that I will discuss the basics of quantum computing and exemplify them by simulating the two quantum algorithms mentioned before, using Mathematica®. The actual implementation of these algorithms will be described in some detail. Some of the figures used in the text (for instance, diagrams of quantum circuits) have been taken from Wikipedia and checked with other sources (they can be used under a Creative Commons Attribution-Share Alike license). The rest of the figures have been drawn by the author with Mathematica®.

2 Quantum Physics

At the end of the XIX century physicists thought that they had discovered almost everything, that they knew all the important secrets of the universe. However, they were wrong. In the earlier years of the XX century, a new field was born: quantum physics. Many of the best physicists and mathematicians of that time, like Planck, Einstein, Schrödinger, Heisenberg, Dirac, Bohr or Von Neumann, among others, worked in this field. As they did they built one of the most mysterious theories in physics. Despite its particularities and its many non-intuitive concepts and ideas, this field has been able to explain, in the most satisfactory way, a huge number of physical phenomena. In fact we do not know of a single physical phenomenon which is not compatible with quantum mechanics.

In order to clarify the concepts of quantum physics, I will try to compare them, when possible, with related classical ideas. However, some are completely new so it is impossible to find something similar to them within classical mechanics.

2.1 Physical Systems

2.1.1 Classical Mechanics

Let us consider a classical mechanical system. We will be able to calculate precisely how it behaves if we know its initial state and its evolution law. Also with this we can compute any observable of the system and compare it with the experimental empirical results. An observable is a function of the variables describing the motion of the system.

If we prepare several identical experiments and measure one observable, we always obtain the same result. As we will mention later this is no longer true for quantum mechanical systems.

To be more specific, I am going to consider an easy example, the point particle. Its state is defined by its position $\mathbf{x} = (x, y, z)$ and its linear momentum $\mathbf{p} = (p_x, p_y, p_z)$ at a given instant of time. If we know these six variables we can compute any function of them $F(\mathbf{x}, \mathbf{p})$. These functions are the observables.

Some well known observables could be the angular momentum, the energy or the velocity.

The last essential thing to consider is how the system evolves. For this in classical mechanics we have the following equations:

$$\frac{\mathrm{d}}{\mathrm{dt}}\begin{pmatrix}\mathbf{x}\\ \mathbf{p}\end{pmatrix} = \begin{pmatrix}\mathbf{p}/m\\ -\nabla V(\mathbf{x})\end{pmatrix}. \tag{1}$$

To solve them we need to specify the initial data for the system (initial state):

$$\begin{pmatrix}\mathbf{x}(0)\\ \mathbf{p}(0)\end{pmatrix} = \begin{pmatrix}\mathbf{x}_0\\ \mathbf{p}_0\end{pmatrix} \tag{2}$$

These equations give us the evolution laws. Their solutions define the state of the system at any instant of time.

2.1.2 Quantum Mechanics

When we move to the quantum world, we keep some of the elements of classical mechanics such as states, observables and evolution laws. However, their mathematical representation is going to be radically different.

The single most important difference between classical and quantum mechanics is the fact that the outcomes of the experiments carried out on identically prepared quantum systems, may differ from one measurement to another. The theory only allows us to compute the probabilities of the possible results of physical measurements.

A way to interpret this fact is by considering that quantum physics refers to ensembles and not to individual systems[1]. We cannot predict the result of a single experiment—even if we have carried out an identical one just before—because, generically, the result is not going to be the same. However, quantum mechanics tells us how we can compute, in principle, the probabilities of its possible outcomes. The frequencies of the results obtained when the experiment is carried out many times will be described by these probabilities.

Einstein's famous quote "God does not play dice" was a provocative way to express this idea. He believed that the probabilistic interpretation could not be applied to a single system[2], that the quantum theory was incomplete. However until now, it seems that God, in the quantum world does play dice.

The state in quantum mechanics allows us to compute the probabilities that the observables take certain values. The mathematical objects used to represent states, are vectors in a Complex Hilbert space $\mathcal{H}$. The Hilbert space somehow plays the role of the $\mathbb{R}^6$ space used in classical mechanics to describe the motion of a point particle. The notation used for these vectors in quantum physics is $|\psi\rangle$. In this notation the "conjugate transpose vector" (actually, an element in $\mathcal{H}^*$, the dual of $\mathcal{H}$) will be denoted as $\langle\psi|$. This is going to be useful later.

[1] This is a somewhat controversial issue related to the interpretation of quantum mechanics.

[2] Actually, Einstein was probably the most visible supporter of the ensemble interpretation of quantum mechanics mentioned before.

Although the Hilbert spaces necessary to fully describe a quantum system are of infinite dimension, for the examples that we are studying in this TFG it suffices to use the complex finite dimensional spaces $\mathbb{C}^n$. This Hilbert space has the standard inner product $\langle u|v\rangle$ defined as

$$\langle u|v\rangle = \sum_{k=1}^{n} \overline{u}_k v_k \quad u = (u_1, \ldots, u_n), v = (v_1, \ldots, v_n), \tag{3}$$

and a norm defined as the inner product of a vector times itself:

$$||\psi||^2 = \langle \psi|\psi\rangle > 0. \tag{4}$$

This norm will always be different from zero unless the vector is 0.

After introducing states, the next step is to appropriately define the observables. In the mathematical descriptions below they are going to be represented by A. In the quantum setting, they are not simple functions as before. Now they are self-adjoint linear operators. They are matrices in $\mathcal{H}$

$$A : \mathcal{H} \to \mathcal{H}. \tag{5}$$

They satisfy

$$A^\dagger = A \quad \dagger = *\mathrm{t}, \tag{6}$$

where $\dagger$ denotes the transposition and complex conjugation. The most important properties of self-adjoint linear operators are:
-All the eigenvalues are real numbers.
-The eigenvectors corresponding to different eigenvalues are orthogonal.
-They define an *spectral decomposition of the identity*.

The eigenvalues represent the different results that we can obtain when measuring an observable. For each eigenvalue we have some associated eigenvectors[3]. By normalizing these eigenvectors we can build an orthonormal basis for the Hilbert space and write states (vectors in the Hilbert space) as linear combinations of vectors in the basis. Depending on the observable that we are diagonalizing we generically get different bases. We can write a state $|\psi\rangle$ as a linear combination of the eigenvectors ($|a_1\rangle, |a_2\rangle, \ldots, |a_n\rangle$) of a self-adjoint operator A ($A|a_k\rangle = a_k|a_k\rangle$):

$$|\psi\rangle = \lambda_1|a_1\rangle + \lambda_2|a_2\rangle + ... + \lambda_n|a_n\rangle = \sum_{k=1}^{n} \lambda_k|a_k\rangle \tag{7}$$

The square of the module (remember that we are using complex numbers) of the coefficients $\lambda_1, \lambda_2, ..., \lambda_n$ represents the probability of finding the basis state $|a_k\rangle$ when measuring the observable A.

$$|\lambda_k|^2 = P(|a_k\rangle) \tag{8}$$

[3]Eigenvalues can be degenerate.

A simple way to determine the coefficients λ_k in (7) is by multiplying both sides of the equation by one of the eigenvectors in the basis.

$$\langle a_j|\psi\rangle = \sum_{k=1}^{n} \lambda_k \langle a_j|a_k\rangle = \lambda_k \tag{9}$$

because the inner product of the orthonormal eigenvectors satisfies:

$$\langle a_j|a_k\rangle = \delta_{jk} := \begin{cases} 1 & k = j \\ 0 & k \neq j \end{cases} . \tag{10}$$

We can then write

$$|\psi\rangle = \sum_{k=1}^{n} |a_k\rangle\langle a_k|\psi\rangle, \tag{11}$$

and, hence,

$$\sum_{i=1}^{n} |x_i\rangle\langle x_i| = \mathbb{I}_n \tag{12}$$

i.e. is the identity matrix in $\mathbb{C}^n$. The coefficient $\langle a_j|\psi\rangle$ is the "wave function" in this finite-dimensional example[4].

We also have the spectral decomposition

$$\sum_{k=1}^{n} a_k |a_k\rangle\langle a_k| = A\,. \tag{13}$$

To compute the probabilities as mentioned before, we have to take the square of the wave function. As everything is normalized we get probabilities in the range of 0 to 1. Notice that the normalization condition implies that these probabilities add up to one.

For example, the probability of "finding" a quantum particle in the range of position from a to b can be calculated as:

$$P_[a, b] = \int_a^b |\psi(x)|^2 dx. \tag{14}$$

If the interval is very small the expression can be approximated as:

$$P_{\triangle x} = |\psi(x)|^2 \triangle x. \tag{15}$$

The wave function of the linear momentum is:

$$\widehat{P}\psi(x) = i\hbar \frac{\partial \psi}{\partial x}, \tag{16}$$

and the probabilities will be obtained in a similar way.

Measuring in quantum physics is a very complex task. Usually, the act of measuring changes the state of the system that you are measuring.

[4]The wave function in the position representation is usually written as $\psi(x)$. Here the function is $\{a_1, \ldots, a_n\} \to \mathbb{C}$.

Compatible observables in quantum mechanics are represented by commuting self-adjoint operators. The measurement of one of them does not influence the result of the other: for example the position along the x direction $\widehat{X}$ and the position along the y axis direction $\widehat{Y}$ can be simultaneously measured with arbitrary precision and the dispersion (the range of possibilities of the measurement of the particle) of both measurements can simultaneously be very small. It is straightforward to check the commutation of $\widehat{X}$ and $\widehat{Y}$:

$$\widehat{X}\widehat{Y}\psi = xy\psi = yx\psi = \widehat{Y}\widehat{X}\psi \rightarrow [\widehat{X},\widehat{Y}] := \widehat{X}\widehat{Y} - \widehat{Y}\widehat{X} = 0\,. \tag{17}$$

An analogous thing happens with the components of the quantum linear momentum $\widehat{\mathbf{P}}$.

When observables do not commute (i.e. they are not compatible) we have a very interesting situation that does not apply to the classical world. Here is where Heisenberg's uncertainty principle appears. This principle establishes an inequality involving the standard deviations of incompatible observables. A typical example is the measurement of position and momentum, on one state (ψ). The uncertainty principle will look like:

$$(\triangle_\psi \widehat{X})(\triangle_\psi \widehat{P}) \geq \frac{\hbar}{2} \tag{18}$$

where $(\triangle_\psi \widehat{X})^2 := \langle\psi|\widehat{X}^2|\psi\rangle - \langle\psi|\widehat{X}|\psi\rangle^2$ and$(\triangle_\psi \widehat{P})^2 := \langle\psi|\widehat{P}^2|\psi\rangle - \langle\psi|\widehat{P}|\psi\rangle^2$ represent the dispersions of measurements of position and momenta in the state $|\psi\rangle$. This means that if the dispersion of the measurements of one observable is small the dispersion in the other will be large.

It is possible to see that this is a consequence of the fact that the operators $\widehat{X}$ and $\widehat{P}$ do not commute. Let us check this by considering $\widehat{X}\widehat{P}\psi$ and $\widehat{P}\widehat{X}\psi$. If they are equal, they commute and therefore they will be compatible observables. However we will see that they do not.

First we expand both expressions:

$$\widehat{X}\widehat{P}\psi = \widehat{X}(\widehat{P}\psi) = \widehat{X}(i\hbar\frac{\partial\psi}{\partial x}) = -i\hbar x\frac{\partial\psi}{\partial x} \tag{19}$$

$$\widehat{P}\widehat{X}\psi = \widehat{P}(\widehat{X}\psi) = \widehat{P}(x\psi) = -i\hbar x\frac{\partial x\psi}{\partial x} = -i\hbar\psi - i\hbar\psi x\frac{\partial\psi}{\partial x} \tag{20}$$

With them we can easily compute the commutator:

$$[\widehat{X},\widehat{P}] = \widehat{X}\widehat{P}\psi - \widehat{P}\widehat{X}\psi = -i\hbar x\frac{\partial\psi}{\partial x} + i\hbar\psi + i\hbar\psi x\frac{\partial\psi}{\partial x} = i\hbar\psi \tag{21}$$

As we can see this is not equal to 0.

Finally, the evolution law in quantum physics is given by the Schrödinger equation,

$$i\hbar\frac{\mathrm{d}}{\mathrm{dt}}|\psi\rangle = \widehat{H}|\psi\rangle \tag{22}$$

$$|\psi(0)\rangle = |\psi_0\rangle \tag{23}$$

where $\widehat{H}$ is the observable known as the Hamiltonian of the system. As we can see it generates time evolution.

2.2 Angular Momentum

In classical physics the angular momentum is a very important observable, measurable and calculable as we have explained before, $\mathbf{L} = \mathbf{X} \times \mathbf{P}$. However, in quantum physics the angular momentum has, as everything, some extra features. It consits of two different parts, $\widehat{\mathbf{L}}$ and $\widehat{\mathbf{S}}$ called orbital angular momentum and spin, respectively. The sum is what we call the total angular momentum and is denoted by $\widehat{\mathbf{J}}$.

$$\widehat{\mathbf{J}} = \widehat{\mathbf{L}} + \widehat{\mathbf{S}} \tag{24}$$

The first part $\widehat{\mathbf{L}}$ can be obtained from the classical expression by replacing the position and momentum by their quantum counterparts. The classical angular momentum is ($\mathbf{P} = m\mathbf{v}$),

$$\mathbf{L} = \mathbf{r} \times m\mathbf{v}\,, \tag{25}$$

and its quantum analog follows the same pattern

$$\widehat{\mathbf{L}} = \widehat{\mathbf{X}} \times \widehat{\mathbf{P}}\,. \tag{26}$$

The spin part $\widehat{\mathbf{S}}$ is inherent and intrinsic to the quantum particles. The spin was discovered in an experiment carried out by Otto Stern and Walther Gerlach. The experiment consisted on sending a beam of silver atoms through a magnetic field in order to deflect them and observe their deviation. The expected result was a continuous distribution of atoms because their magnetic momenta were randomly oriented (see figure).

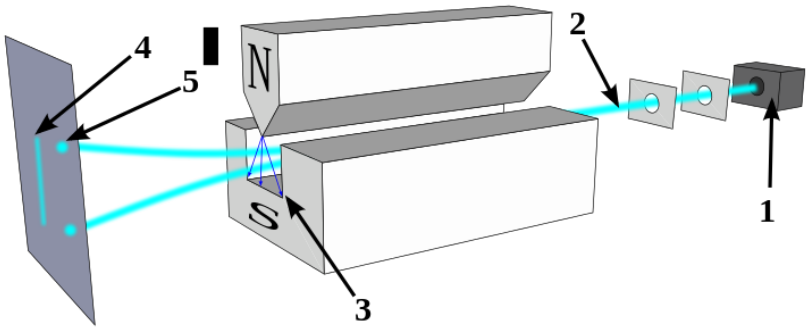


Figure 1: Stern-Gerlach experiment [3]

However the result was different and rather surprising. They found two different spots where all the particles ended up. The interpretation of this result is that the

angular momentum is quantized although, in the particular case of this experiment it was necessary to introduce the concept of spin to completely account for the observed result. With this discovery finally the properties of the some subatomic particles like the electrons were fully described. Many subatomic particles have spin. Together with mass, the spin is one the basic properties of the elementary particles. The quantum spin has no classical analog. Subatomic particles can be classified according to their spins in two categories, fermions (if the spin is half integer, $\frac{1}{2}$, $\frac{3}{2}$, ...) and bosons (if the spin is integer 0, 1, 2,...). Their physical behaviours are significantly different.

At variance with the case of the position or momenta observables the components of the angular momentum *do not* commute (although the orbital and spin parts do commute). This applies both to the orbital and spin parts. The commutation rules for the components of $\mathbf{L}$ are:

$$[\widehat{L}_x, \widehat{L}_y] = i\hbar\widehat{L}_z \tag{27}$$
$$[\widehat{L}_y, \widehat{L}_z] = i\hbar\widehat{L}_x \tag{28}$$
$$[\widehat{L}_z, \widehat{L}_x] = i\hbar\widehat{L}_y \tag{29}$$

Similarly for the spin components:

$$[\widehat{S}_x, \widehat{S}_y] = i\hbar\widehat{S}_z \tag{30}$$
$$[\widehat{S}_y, \widehat{S}_z] = i\hbar\widehat{S}_x \tag{31}$$
$$[\widehat{S}_z, \widehat{S}_x] = i\hbar\widehat{S}_y \tag{32}$$

Generically the spin commutation rule is:

$$[\widehat{S}_i, \widehat{S}_j] = i\hbar\sum_{k=1}^{3}\epsilon_{ijk}\widehat{S}_k\,, \tag{33}$$

where ϵ_{ijk} is the usual totally antisymmetric symbol satisfying $\epsilon_{123} = 1$. For the purpose of this work, we only need to consider the spin part.

2.2.1 Spin $\frac{1}{2}$

Now we go into the different spin observables. The spin operator for $s = \frac{1}{2}$ can be written in terms of the Pauli matrices as:

$$\widehat{\mathbf{S}} = \frac{\mathbf{1}}{\mathbf{2}}\hbar\boldsymbol{\sigma}\mathbf{,} \tag{34}$$

where each matrix is associated with a direction.

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{35}$$

The spin in the direction defined by the unitary vector $\mathbf{u} = (u_x, u_y, u_z)$ is

$$S_{\mathbf{u}} = \frac{1}{2}\hbar(\mathbf{u}\sigma) = \frac{1}{2}\hbar(u_x\sigma_x + u_y\sigma_y + u_z\sigma_z) \tag{36}$$

It is important to know the main properties of the Pauli matrices: They are self-adjoint (otherwise they cannot be observables), their eigenvalues are $+1$ and -1 and Finally their eigenvectors for each eigenvalue and in each direction are:

$$|+\rangle_x = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad |-\rangle_x = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}, \tag{37}$$

$$|+\rangle_y = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ i \end{pmatrix} \quad |-\rangle_y = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -i \end{pmatrix}, \tag{38}$$

$$|+\rangle_z = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |-\rangle_z = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{39}$$

When we square them, we get the two dimensional identity matrix ($\mathbb{I}_2$).

$$\sigma_x^2 = \sigma_y^2 = \sigma_z^2 = \mathbb{I}_2 \tag{40}$$

Substituting the Pauli matrices in equation 36, gives us with the general spin observable (for any direction $\mathbf{u}$). For spin $\frac{1}{2}$:

$$S_{\mathbf{u}} = \frac{1}{2}\hbar(u_x\sigma_x + u_y\sigma_y + u_z\sigma_z) = \frac{1}{2}\hbar\begin{pmatrix} u_z & u_x - iu_y \\ u_x + iu_y & -u_z \end{pmatrix} \tag{41}$$

The eigenvalues of this observable are $+\frac{\hbar}{2}$ and $-\frac{\hbar}{2}$. Their associated eigenvectors are

$$|\psi_+\rangle_{\mathbf{u}} = \begin{pmatrix} \dfrac{u_x - iu_y}{1 - u_z} \\ \\ 1 \end{pmatrix} \tag{42}$$

$$|\psi_-\rangle_{\mathbf{u}} = \begin{pmatrix} \dfrac{-u_x + iu_y}{1 + u_z} \\ \\ 1 \end{pmatrix} \tag{43}$$

When we substitute in $\mathbf{u}$ a specific direction we get the corresponding eigenvectors. However this general expression does not work in all the possible cases. When we try to substitute $\mathbf{u} = (0, 0, 1)$ we realize that is impossible to compute the eigenvectors because the denominator of their first component is 0. To avoid this problem, in the $\widehat{Z}$ direction, we use directly the σ_z Pauli matrix.

$$\widehat{S}_z = \frac{1}{2}\hbar\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{44}$$

Its eigenvalues are $+\frac{\hbar}{2}$ and $-\frac{\hbar}{2}$ and its eigenvectors

$$|+\rangle_z = \frac{\hbar}{2}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |-\rangle_z = \frac{\hbar}{2}\begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{45}$$

This result is going to be very important later. For simplicity the notation will be changed a little bit, writing $|+\rangle_z$ and $|-\rangle_z$ as $|0\rangle$ and $|1\rangle$.

In order to built orthonormal bases with the preceding eigenvectors we have to normalize them. This can be done by computing the modules (remember that we are working with complex numbers so for the module we have to take the conjugate) $\langle +|+\rangle_u = \frac{2}{1-u_z}$ and $\langle -|-\rangle_u = \frac{2}{1+u_z}$. By dividing the eigenvectors by their corresponding modules, we get:

$$|+\rangle_{\mathbf{u}} = \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{u_x - iu_y}{\sqrt{1-u_z}} \\ \\ \sqrt{1-u_z} \end{pmatrix} \tag{46}$$

$$|-\rangle_{\mathbf{u}} = \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{-u_x + iu_y}{\sqrt{1+u_z}} \\ \\ \sqrt{1+u_z} \end{pmatrix} \tag{47}$$

We can write a generic spin state

$$|\psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix} \tag{48}$$

in the basis $\{|+\rangle_u, |-\rangle_u\}$ as:

$$|\psi\rangle = \lambda_a |+\rangle_{\mathbf{u}} + \lambda_b |-\rangle_{\mathbf{u}} = \lambda_a \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{u_x - iu_y}{\sqrt{1-u_z}} \\ \\ \sqrt{1-u_z} \end{pmatrix} + \lambda_b \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{-u_x + iu_y}{\sqrt{1+u_z}} \\ \\ \sqrt{1+u_z} \end{pmatrix} \tag{49}$$

The coefficients λ_a and λ_b are given by:

$$\lambda_a = \langle +|\psi\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{u_x + iu_y}{\sqrt{1-u_z}} & \sqrt{1-u_z} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{\sqrt{2}} \left(\frac{u_x + iu_y}{\sqrt{1-u_z}} a + \sqrt{1-u_z} b \right) \tag{50}$$

$$\lambda_b = \langle -|\psi\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{-u_x - iu_y}{\sqrt{1+u_z}} & \sqrt{1+u_z} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{\sqrt{2}} \left(\frac{-u_x - iu_y}{\sqrt{1+u_z}} a + \sqrt{1+u_z} b \right) \tag{51}$$

These coefficients are very useful. The square of their modules are the probability of getting the spin oriented in the $\mathbf{u}$ or $-\mathbf{u}$ directions. Once we have computed the coefficients, we can directly check the that 49 and 50 are valid:

$$\begin{aligned} |\psi\rangle =& \lambda_a |+\rangle_{\mathbf{u}} + \lambda_b |-\rangle_{\mathbf{u}} \\ =& \frac{1}{2} \left(\frac{u_x + iu_y}{\sqrt{1-u_z}} a + \sqrt{1-u_z} b \right) \begin{pmatrix} \frac{u_x - iu_y}{\sqrt{1-u_z}} \\ \\ \sqrt{1-u_z} \end{pmatrix} \\ &+ \frac{1}{2} \left(\frac{-u_x - iu_y}{\sqrt{1+u_z}} a + \sqrt{1+u_z} b \right) \begin{pmatrix} \frac{-u_x + iu_y}{\sqrt{1+u_z}} \\ \\ \sqrt{1+u_z} \end{pmatrix} \\ =& \begin{pmatrix} a \\ b \end{pmatrix} \end{aligned}$$

We check now equation 11.

$$\begin{aligned}\mathbb{I}_2 &= \sum_k |u_k\rangle\langle u_k| = |+\rangle\langle +| + |-\rangle\langle -| \\ &= \frac{1}{2}\begin{pmatrix} \frac{u_x - iu_y}{\sqrt{1-u_z}} \\ \sqrt{1-u_z} \end{pmatrix}\left(\frac{u_x + iu_y}{\sqrt{1-u_z}} \ \sqrt{1-u_z}\right) \\ &+ \frac{1}{2}\begin{pmatrix} \frac{-u_x + iu_y}{\sqrt{1+u_z}} \\ \sqrt{1+u_z} \end{pmatrix}\left(\frac{-u_x - iu_y}{\sqrt{1+u_z}} \ \sqrt{1+u_z}\right)\end{aligned}$$

We can also see that equation 13 holds

$$\begin{aligned}S_{\mathbf{u}} &= \hbar\sum_k \lambda_k |u_k\rangle\langle u_k| = \frac{\hbar}{2}(|+\rangle\langle +| - |-\rangle\langle -|) \\ &= \frac{\hbar}{2}\begin{pmatrix} \frac{u_x - iu_y}{\sqrt{1-u_z}} \\ \sqrt{1-u_z} \end{pmatrix}\left(\frac{u_x + iu_y}{\sqrt{1-u_z}} \ \sqrt{1-u_z}\right) \\ &- \frac{\hbar}{2}\begin{pmatrix} \frac{-u_x + iu_y}{\sqrt{1+u_z}} \\ \sqrt{1+u_z} \end{pmatrix}\left(\frac{-u_x - iu_y}{\sqrt{1+u_z}} \ \sqrt{1+u_z}\right).\end{aligned}$$

With these examples, I hope that the quantum concepts introduced so far are much more clear.

In classical mechanics measuring the angular momentum along different directions does not pose any fundamental problem. However, in quantum mechanics the situation is quite different. As commented before if two observables do not commute it is impossible to simultaneously measure them. In the case of spin, this means that we cannot measure any two of the components $\widehat{S}_x, \widehat{S}_y$ and $\widehat{S}_z$ simultaneously. This does not mean that it is impossible to find an observable that commutes with one of the $\widehat{S}_x, \widehat{S}_y$ or $\widehat{S}_z$. In fact the observable $\widehat{\mathbf{S}}^2$ commutes with all of them as can be shown in general

$$[\widehat{S}_x, \widehat{\mathbf{S}}^2] = [\widehat{S}_x, \widehat{S}_x^2 + \widehat{S}_y^2 + \widehat{S}_z^2] = [\widehat{S}_x, \widehat{S}_x^2] + [\widehat{S}_x, \widehat{S}_y^2] + [\widehat{S}_x, \widehat{S}_z^2] \tag{52}$$

Now we prove that the sum of this commutators is 0 we prove that the initial commutator is 0.

$$[\widehat{S}_x, \widehat{S}_x^2] = \widehat{S}_x\widehat{S}_x^2 - \widehat{S}_x^2\widehat{S}_x = \widehat{S}_x^3 - \widehat{S}_x^3 = 0 \tag{53}$$

$$[\widehat{S}_x, \widehat{S}_y^2] = \widehat{S}_x\widehat{S}_y^2 - \widehat{S}_y^2\widehat{S}_x = \widehat{S}_y[\widehat{S}_x, \widehat{S}_y] - [\widehat{S}_y, \widehat{S}_x]\widehat{S}_y = \widehat{S}_y i\hbar\widehat{S}_z + i\hbar\widehat{S}_z\widehat{S}_y \tag{54}$$

$$[\widehat{S}_x, \widehat{S}_z^2] = \widehat{S}_x\widehat{S}_z^2 - \widehat{S}_z^2\widehat{S}_x = \widehat{S}_z[\widehat{S}_x, \widehat{S}_z] - [\widehat{S}_z, \widehat{S}_x]\widehat{S}_z = -\widehat{S}_z i\hbar\widehat{S}_y - i\hbar\widehat{S}_y\widehat{S}_z \tag{55}$$

$$[\widehat{S}_x, \widehat{\mathbf{S}}^2] = 0 + \widehat{S}_y i\hbar\widehat{S}_z + i\hbar\widehat{S}_z\widehat{S}_y - \widehat{S}_z i\hbar\widehat{S}_y - i\hbar\widehat{S}_y\widehat{S}_z = 0 \tag{56}$$

By measuring this observable we can find out the spin of a quantum particle.

In the case of a spin $\frac{1}{2}$ particle it is straightforward to see that $\widehat{\mathbf{S}}^2$ is proportional to $\mathbb{I}_2$

$$\widehat{S}^2 = \widehat{S}_x^2 + \widehat{S}_y^2 + \widehat{S}_z^2 = \frac{3}{4}\hbar^2\mathbb{I}_2 = \frac{1}{2}(\frac{1}{2}+1)\hbar^2\mathbb{I}_2\,, \tag{57}$$

and hence it trivially commutes with $\widehat{S}_x, \widehat{S}_y$ and $\widehat{S}_z$.

For the sake of completeness we give now some expressions corresponding to higher spins

2.2.2 Spin 1

With the spin 1 we follow the same pattern. However, this time we do not use the Pauli Matrices, but other (3×3) matrices with similar properties (self-adjoint, real eigenvalues, real eigenvectors).

$$\widehat{S}_x = \frac{\hbar}{\sqrt{2}}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \widehat{S}_y = \frac{\hbar}{\sqrt{2}}\begin{pmatrix} 0 & -i & 0 \\ i & 0 & -i \\ 0 & i & 0 \end{pmatrix} \widehat{S}_z = \hbar\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix} \tag{58}$$

Their eigenvalues are $+\hbar, 0, -\hbar$ and the corresponding eigenvectors are:

$$|+\rangle_x = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{\sqrt{2}} \\ \frac{1}{2} \end{pmatrix}, |0\rangle_x = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}, |-\rangle_x = \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{\sqrt{2}} \\ \frac{1}{2} \end{pmatrix}. \tag{59}$$

$$|+\rangle_y = \begin{pmatrix} -\frac{1}{2} \\ -\frac{i}{\sqrt{2}} \\ \frac{1}{2} \end{pmatrix}, |0\rangle_y = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}, |-\rangle_y = \begin{pmatrix} -\frac{1}{2} \\ \frac{i}{\sqrt{2}} \\ \frac{1}{2} \end{pmatrix}. \tag{60}$$

$$|+\rangle_z = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, |0\rangle_z = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, |-\rangle_z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{61}$$

We have now

$$\widehat{\mathbf{S}}^2 = 2\hbar^2\mathbb{I}_3 = 1(1+1)\hbar^2 I_3. \tag{62}$$

2.2.3 Spin s

Following these examples we can see how the other spins are going to work. It is always going to be proportional to matrices of dimension $2s+1$. The eigenvalues of the spin components are $-\hbar s, -\hbar(s-1), \ldots, \hbar s$. The squared spin observable is

$$\widehat{\mathbf{S}}^2 = s(s+1)\hbar^2\mathbb{I}_{2s+1}\,. \tag{63}$$

2.3 Compound Systems

2.3.1 Classical Compound Systems

In order to describe classical compound systems one must combine in a certain way their configuration or phase spaces. This is straightforward because it suffices to take their cartesian product. For example to describe the dynamics of one point particle moving in a straight line we need to know its position and momentum

$$\begin{pmatrix} X \\ P \end{pmatrix} \in \mathbb{R}^2 \,. \tag{64}$$

If we have two particles we can describe their motion by giving four numbers: two to specify the positions an another two for the momenta

$$\begin{pmatrix} X_1 \\ P_1 \\ X_2 \\ P_2 \end{pmatrix} \in \mathbb{R}^4 = \mathbb{R}^2 \times \mathbb{R}^2 \,. \tag{65}$$

The generalization for n particles is obvious. In general we need $2n$ quantities to completely specify the instantaneous state of the system.

2.3.2 Quantum Compound Systems

We have just seen that the configuration (phase) space of a compound classical system is the cartesian product of the configuration spaces of its subsystems. It would be natural, then, to think that the Hilbert space of a compound quantum system would be just the "cartesian product" of the Hilbert spaces of the individual subsystems. A profound and far reaching feature of quantum mechanics is the fact that this is not the case: in order to describe a compound quantum system one must use the *tensor product* of the Hilbert spaces of the subsystems. The quintessential quantum phenomenon known as entanglement is related to this fact. As the tensor product construction plays a central role in quantum computing we describe it now in some detail.

The rigorous definition of the tensor product is beyond the scope of this TFG (an probably not very illuminating) but it is relatively straightforward to explain how it works. The tensor product of

$$|u\rangle = \begin{pmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ u_m \end{pmatrix} \in \mathcal{H}_1 \qquad |v\rangle = \begin{pmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ \cdot \\ v_n \end{pmatrix} \in \mathcal{H}_2 \tag{66}$$

is denoted as $|u\rangle \otimes |v\rangle$ and is given by

$$|u\rangle \otimes |v\rangle = \begin{pmatrix} u_1 v_1 \\ u_1 v_2 \\ . \\ . \\ . \\ u_1 v_n \\ u_2 v_1 \\ u_2 v_2 \\ . \\ . \\ . \\ u_m v_n \end{pmatrix} . \tag{67}$$

These vectors are the elements of $\mathcal{H}_1 \otimes \mathcal{H}_2$, the tensor product of the Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$. The scalar product of the vectors $|u_1\rangle \otimes |v_1\rangle$ and $|u_2\rangle \otimes |v_2\rangle$ is $\langle u_1|u_2\rangle\langle v_1|v_2\rangle$.

The tensor product can be generalized in straightforward way to other objects (such as dual vectors or matrices). For instance, $|u\rangle \otimes \langle v|$ is given by

$$|u\rangle \otimes \langle v| = \begin{pmatrix} u_1\overline{v}_1 & u_1\overline{v}_2 & . & . & . & u_1\overline{v}_n \\ u_2\overline{v}_1 & u_2\overline{v}_2 & . & . & . & u_2\overline{v}_n \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ u_m\overline{v}_0 & u_m\overline{v}_1 & . & . & . & u_m\overline{v}_n \end{pmatrix} , \tag{68}$$

where we have to remember that $\langle v| = (\,\overline{v}_1\,\overline{v}_2 \ldots \overline{v}_n\,)$.

In this TFG we will use the following tensor products of Hilbert spaces. For one particle of spin $\frac{1}{2}$ we have:

$$\mathcal{H} = \mathbb{C}^2 \tag{69}$$

For two spin $\frac{1}{2}$ particles we have:

$$\mathcal{H} = \mathbb{C}^2 \otimes \mathbb{C}^2 \tag{70}$$

While for N spin $\frac{1}{2}$ particles we have:

$$\mathcal{H} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes ... \otimes \mathbb{C}^2 =: (\mathbb{C}^2)^{\otimes n} . \tag{71}$$

The elements of $(\mathbb{C}^2)^{\otimes n}$ have 2^n components.

We have just described the elements in the tensor product of Hilbert spaces. In the following subsection we will some interesting operators in the Hilbert space of a system consisting of two spin $\frac{1}{2}$ particles.

2.3.3 System of two spin $\frac{1}{2}$ particles

In the following we are going to define spin operators in $\mathbb{C}^2 \otimes \mathbb{C}^2$, the Hilbert space of two $\frac{1}{2}$ particles. From here on we will take $\hbar = 1$.

$$\widehat{\mathbf{S}} := \widehat{\mathbf{S}}_\mathbf{1} \otimes \mathbb{I}_\mathbf{2} + \widehat{\mathbf{S}}_\mathbf{2} \otimes \mathbb{I}_\mathbf{2} = \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\sigma} \otimes \mathbb{I}_\mathbf{2} + \mathbb{I}_\mathbf{2} \otimes \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\sigma}\,. \tag{72}$$

This operator acts on $|u\rangle \otimes |v\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2$ in the following way.

$$\widehat{\mathbf{S}}(|u\rangle \otimes |v\rangle) = (\widehat{\mathbf{S}}_1|u\rangle) \otimes |v\rangle + |u\rangle \otimes (\widehat{\mathbf{S}}_2|v\rangle)\,. \tag{73}$$

For the x direction:

$$\widehat{S}_x = \sigma_x \otimes \mathbb{I}_2 + \mathbb{I}_2 \otimes \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} =$$

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{74}$$

This matrix represents the total spin observable of the system in the **x** direction.

Analogously for the y direction:

$$\widehat{S}_y = \sigma_y \otimes \mathbb{I} + \mathbb{I} \otimes \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} =$$

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & -i & 0 \\ 0 & 0 & 0 & -i \\ i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 0 & -i & -i & 0 \\ i & 0 & 0 & -i \\ i & 0 & 0 & -i \\ 0 & i & i & 0 \end{pmatrix} \tag{75}$$

This matrix represents the total spin observable of the system in the **y** direction.

Finally for the z direction:

$$\widehat{S}_z\sigma_z \otimes \mathbb{I}_2 + \mathbb{I}_2 \otimes \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} =$$

$$\frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \end{pmatrix} \tag{76}$$

This matrix represents the total spin observable of the system in the **z** direction.

Another important observable is $\widehat{\mathbf{S}}^2 = \widehat{\mathbf{S}}_x^2 + \widehat{\mathbf{S}}_y^2 + \widehat{\mathbf{S}}_z^2$. Taking into account

$$\widehat{S_x}^2 = \frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \tag{77}$$

$$\widehat{S_y}^2 = \frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \tag{78}$$

$$\widehat{S_z}^2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{79}$$

we find

$$\widehat{\mathbf{S}}^2 = S_x^2 + S_y^2 + S_z^2 = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}. \tag{80}$$

It is straightforward to see that the commutation rules

$$[\widehat{S}_i, \widehat{S}_j] = i\sum_{k=1}^{3} \epsilon_{ijk}\widehat{S}_k \tag{81}$$

and

$$[\widehat{\mathbf{S}}^2, \widehat{S}_i] = 0 \tag{82}$$

hold.

The eigenvalues of this observable are 2 and 0. The corresponding normalized eigenvectors are

$$|v_+\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}; |v_o\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}; |v_-\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \tag{83}$$

for the eigenvalue $s = 2 = 1(1+1)$ and

$$|w\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ 1 \\ -1 \\ 0 \end{pmatrix}. \tag{84}$$

for the eigenvalue $s = 0$.

In this basis the observables $\widehat{S}_x$, $\widehat{S}_y$, $\widehat{S}_z$ and $\widehat{\mathbf{S}}^2$ take the form

$$\widehat{S}_x \to \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{85}$$

$$\widehat{S}_y \to \begin{pmatrix} 0 & \frac{-i}{\sqrt{2}} & 0 & 0 \\ \frac{i}{\sqrt{2}} & 0 & \frac{-i}{\sqrt{2}} & 0 \\ 0 & \frac{i}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{86}$$

$$\widehat{S}_x \to \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{87}$$

$$\widehat{\mathbf{S}}_x^2 \to \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} . \tag{88}$$

In the previous matrices we can recognize in block form the spin 1 and spin 0 operators introduced above. In fact the tensor product $\mathbb{C}^2 \otimes \mathbb{C}^2$ can be decomposed as

$$\mathbb{C}^2 \otimes \mathbb{C}^2 = \text{span}\{|v_+\rangle, |v_0\rangle, |v_-\rangle\} \oplus \text{span}\{|\text{w}\rangle\} . \tag{89}$$

This is usually written in the form $\frac{1}{2} \otimes \frac{1}{2} = 1 \oplus 0$ expressing the fact that the composition of two spins $\frac{1}{2}$ gives a spin 1 (triplet) and a spin 0 (singlet).

3 Quantum Computation

The trend on building computers since their invention has been miniaturization (making the internal components like the transistor smaller and smaller). However this cannot continue forever. Even if they could, classical computation has limits: for some problems the size of the computers needed in order to reach a solution is too big. For example we would need a computer with size equal to the number of atoms of the Milky Way galaxy to fully describe the properties of a molecule like caffeine [4]. Other problems, even more complicated, are impossible to solve with the classical computers, no matter their size.

Another constraint is that when the transistors reach the scale of nanometers (we have reach that scale in many science fields), they will start malfunctioning. The quantum properties of electrons make them unpredictable (they escape from their channels due to the duality particle-wave) and we have what is called the tunnel effect.

In order to keep progressing in computer science, we need a change of paradigm. Many scientists believe that this new paradigm will involve quantum computation.

3.1 The Qubit

In classical computations the smallest unit of information is the bit. The bit can take the value of 1 or 0. We need many bits in order to store information. The collections of bits are called bit registers. The elements in charge of transforming these bit registers are the logic gates. These logic gates are modelled by the programs in order to carry on useful computations. The programs do not care about how the bits are actually implemented because they only focus on the value of the bit, 0 or 1.

Quantum computations follow a similar pattern. The smallest unit of information is the qubit. However the qubit is far more powerful than the bit. It takes advantage from the non-intuitive phenomena of the quantum world. Instead of having the "capacity" of just being 1 or 0, the qubit can somehow be both at the same time. This is a consequence of the superposition principle. In fact the qubit can take any normalized "value" in the two dimensional state space $\mathbb{C}^2$ spanned by the orthonormal basis $|1\rangle$ and $|0\rangle$. These normalized states are expressed as linear combinations of both orthonormal vectors with coefficients a_0 and a_1, satisfying $|a_0|^2 + |a_1|^2 = 1$.

$$|\psi\rangle = a_0 |0\rangle + a_1 |1\rangle \tag{90}$$

This can be pictured in the Bloch Sphere. In order to arrive at the Bloch Sphere we have to perform some transformations in the coefficients. As a_0 and a_1 are two complex numbers we have four degrees of freedom:

$$a_0 = \mathrm{Re}(a_0) + i\mathrm{Im}(a_0) = |a_0|e^{i\theta_0} \tag{91}$$

$$a_1 = \mathrm{Re}(a_1) + i\mathrm{Im}(a_1) = |a_1|e^{i\theta_1} \tag{92}$$

However, some of these four degrees of freedom are redundant. The first constraint that we have is normalization.

$$|\mathrm{Re}_{(a_0)}|^2 + |\mathrm{Im}_{(a_0)}|^2 + |\mathrm{Re}_{(a_1)}|^2 + |\mathrm{Im}_{(a_1)}|^2 = 1 \tag{93}$$

One module can be expressed in terms of the other. With this we remove one degree of freedom.

As we can always multiply a state vector in $\mathbb{C}^2$ by an arbitrary phase we can take a_0 to be a real number ($\mathrm{Im}(a_0) = 0$). By doing this we remove a second degree of freedom.

Taking into account the previous two statements we see that we can map physical states in $\mathbb{C}^2$ to the points of the unit sphere $S_2 \in \mathbb{R}^2$ given by $|\mathrm{Re}_{(a_0)}|^2 + |\mathrm{Re}_{(a_1)}|^2 + |\mathrm{Im}_{(a_1)}|^2 = 1$. As we have chosen $\theta_0 = 0$ our coefficients for the Bloch Sphere are going to be:

$$a_0 = \mathrm{Re}(a_0) \tag{94}$$

$$a_1 = \sqrt{(1 - (\mathrm{Re}(a_0))^2)}\; e^{i\theta_1} \tag{95}$$

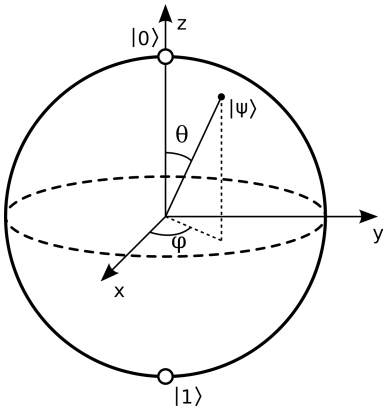


Figure 2: Bloch sphere in spherical coordinates. [6]

As with the classical computer, in order to perform complicated tasks, we will need many qubits (quantum registers) and logic gates. However, this number is usually small compared to the number of bits needed for a similar classical computation. To clarify let us see an example:

Ten bits have 2^{10} possible combinations of zeros and ones. With all these bits we only can represent one number from 0 to 1023. In order to store all the numbers between 0 and 1023 we would need 1024 bit registers. With 10 qubits, by using superposition, we can encode all the numbers between 0 and 1023 at the same time [5].

Quantum computations are carried out by quantum gates acting on qubits. As in the classical computers, the quantum algorithms do not care about how the qubits are modeled.

3.1.1 Physical modeling

There is no standard accepted way to model physical qubits, there are many different solutions. The main problem whith the physical implementation of the

qubits is to avoid decoherence. The qubit is extremely delicate, any interference can decohere it, even the signal that we use to control it. As we increase the number of qubits, the system becomes more and more fragile. The bigger the number of qubits, the more complex the system.

The main solutions are:

- Superconducting quantum computing: The qubit implemented as the state of small superconducting circuits. Many big companies like IBM[5], Google or Intel are researching in this solution.
- Trapped ion computer. The qubits are implemented as internal states of trapped ions.
- Optical lattices where: The qubits are implemented as the internal states of neutral atoms trapped in a optical lattice.
- Quantum dot computer: There are two main approaches:
  - Spin-based: The qubits are encoded in spins of trapped electron.
  - Spatial-based: The qubits are encoded in the position of the electrons.
- Nuclear magnetic resonance of molecules in solutions: The qubits are nuclear spins of the dissolved molecules.

3.1.2 Agreggation of qubits

As we have seen before, we need to work with several qubits in order to carry out non-trivial quantum computations. In order to obtain the state vector for the combined quantum state corresponding to the aggregation of qubits we need to use tensor products. Also, we have to ensure that the result is normalized.

The qubit can be modelled as a spin $\frac{1}{2}$. To built the qubit states, we need to take a orthonormal basis in the Hilbert space. To make it simpler, we are going to choose a basis similar to the eigenvectors of the spin $\frac{1}{2}$ in the z direction ($\widehat{Z}$) (45).

When we normalize them we end up with the following basis:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{96}$$

In Dirac notation we can express them as $|0\rangle$ and $|1\rangle$. A qubit can be in state equal to one of the basis (the qubit could be $|0\rangle$ or $|1\rangle$) or equal to a linear combination of them ($|\psi\rangle = a_0|0\rangle + a_1|1\rangle$).

To get the combined state of two qubits $|\alpha\rangle$ and $|\beta\rangle$ we have to take the tensor product of them: $|\alpha\beta\rangle = |\alpha\rangle \otimes |\beta\rangle$. For instance:

[5]I would like to mention at this point the availability of an online quantum computer where simple quantum programs can be implemented and run.
See https://quantumexperience.ng.bluemix.net/qx/editor

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{97}$$

With three qubits we can write states such as $|101\rangle$:

$$|101\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{98}$$

The procedure works exactly the same for more qubits. As explained before, then the number of components of these state increases exponentially as 2^n, being n the number of qubits.

These are the basis states. We can built superposition states as linear combinations of all the possible basis states of that number of qubits, for example, for three qubits we have eight possible combinations: $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$ and $|111\rangle$.

The most general state for this three qubit system can be written as:

$$\psi = a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|110\rangle + a_7|111\rangle$$

where the a_0, a_1, a_2, a_3, a_4, a_5, a_6 and a_7 are the coefficients ($\in \mathbb{C}$) of the linear combination of the basic states. The normalization condition for them is

$$\sum_{i=1}^{7} |a_i|^2 = 1 \tag{99}$$

When we measure on the 3-qubit system, the result will be given by one of the basis states. However, at intermediate stages of a quantum computation we will usually have superpositions.

3.2 Quantum Gates

Classical logic gates take an array of bits (a bit register) as input and, through logic operations, produce a different bit register as output. Their action can be described by using boolean algebra. Quantum gates work in a similar way. However, boolean algebra is not enough to describe how they work.

Classical logic circuits consist of a number of connected logic gates. They can be schematically represented as shown[6] in figure 3. After introducing an initial input, the successive gates produce outputs that, in turn, become inputs for other

[6] Although their practical implementation in an actual circuit may not conform to the simple schematics shown here.

gates in the circuit until a final result is produced. This can be a single binary digit or a certain number of them.

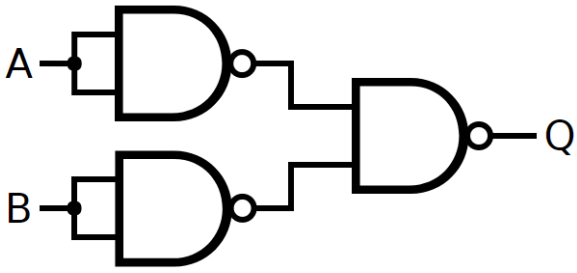


Figure 3: Example of a logic circuit of NAND gates [7]

Quantum gates and circuits are very different. In figure 4 we show a simple example. Each line starting from the left represents a single qubit. The gates, that may act on single qubits or combinations of them, are drawn as boxes or other symbols connecting the qubits involved. Quantum gates correspond to unitary matrices. Unitarity is necessary in order to ensure that probability is conserved through the computation. The operation of a quantum circuit consists of a sequence of unitary transformations $U_1, U_2, \ldots, U_n$, representing quantum gates, acting on the initial quantum state $|\psi_{inital}\rangle$. These transformations are unitary and therefore reversible.

$$|\psi_{final}\rangle = U_n \cdots U_2 \cdot U_1 |\psi_{inital}\rangle \tag{100}$$

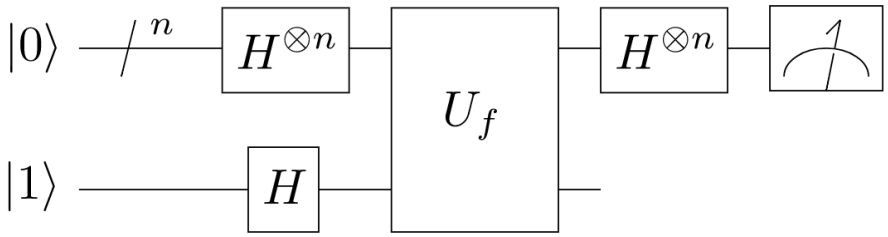


Figure 4: Example of a quantum circuit (Deutsch-Jozsa Algorithm) [8]

During the operation of the quantum circuit the state should not collapse. The power of quantum computation hinges on keeping the entanglement among the different qubits until the computation ends. The state only collapses in the final measurement used to get the result.

I will briefly describe now the main quantum gates used in quantum computers and how they work. I will first describe gates that act on a single qubit, such as the Hadamard and phase shift gates, and later some others that act on two qubits like the CNOT.

3.2.1 The Hadamard Gate $\widehat{H}$

This is one of the most commonly used quantum gates. It is a unitary operator that acts on a single qubit. Its matrix in the basis of $\mathbb{C}^2$ that we are using for a

one-qubit system is [12]

$$\widehat{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} . \tag{101}$$

Acting on each of the one qubit basis states

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} . \tag{102}$$

we get a normalized linear combination (superposition) with coefficients of equal modulus but different relative signs,

$$\widehat{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) ,$$
$$\widehat{H}|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) .$$

In both cases the probabilities of finding the qubit in the $|0\rangle$ or $|1\rangle$ states are the same. We can undo the effect of a Hadamard gate on a state by acting again with it as it is straightforward to see that $\widehat{H}^2 = \mathbb{I}$ (it is *idempotent*).

In a situation when we have several qubits we can have Hadamard operators that act on each of them (while leaving the rest untouched). They can be built by using the identity matrix $\mathbb{I}$, the Hadamard operator acting on a single qubit that we have discussed above, and the tensor product. For instance, for a three qubit system the three Hadamard operators are defined as[10]:

$$\widehat{H_1} = \widehat{H} \otimes \mathbb{I} \otimes \mathbb{I} , \tag{103}$$
$$\widehat{H_2} = \mathbb{I} \otimes \widehat{H} \otimes \mathbb{I} , \tag{104}$$
$$\widehat{H_3} = \mathbb{I} \otimes \mathbb{I} \otimes \widehat{H} . \tag{105}$$

The corresponding matrices in the basis that we have introduced above are

$$\widehat{H_1} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix} ,$$

$$\widehat{H_2} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix} ,$$

$$\widehat{H_3} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix},$$

It is easy to check that these matrices reproduce the action of the Hadamard operators $\widehat{H_1}$, $\widehat{H_2}$ and $\widehat{H_3}$. For instance

$$\widehat{H_1}|100\rangle = H\otimes\mathbb{I}\otimes\mathbb{I}(|1\rangle\otimes|0\rangle\otimes|0\rangle) = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\otimes|0\rangle\otimes|0\rangle = \frac{1}{\sqrt{2}}(|000\rangle - |100\rangle) .$$

We can also check that the preceding matrices commute. This means that if several Hadamard operators act successively, their order does not matter.

Finally, if the Hadamard operators corresponding to all the qubits in our system act on a basis state the resulting vector will have components of the same modulus (a superposition with equal probabilities of finding all the possible qubit states).

A completely analogous discussion applies to systems of any number of qubits.

3.2.2 Phase Shift Gate

The phase shift gate is used to introduce a relative phase shift between the amplitudes without changing the probabilities of finding each basis state. The phase shift matrix for one qubit is [10]:

$$\widehat{R}(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{106}$$

For more qubits, the phase shift gate is defined as above. For example, for the three qubit system we have:

$$\widehat{R_1}(\theta) = \widehat{R}(\theta) \otimes \mathbb{I} \otimes \mathbb{I} \tag{107}$$

$$\widehat{R_2}(\theta) = \mathbb{I} \otimes \widehat{R}(\theta) \otimes \mathbb{I} \tag{108}$$

$$\widehat{R_3}(\theta) = \mathbb{I} \otimes \mathbb{I} \otimes \widehat{R}(\theta) \tag{109}$$

Notice that, for $\theta \neq 0$ the phase shift does not commute with the Hadamard operator. Also notice that, generically, $\widehat{R}(\theta)$ is not idempotent.

The matrix forms of $\widehat{R_1}(\theta)$, $\widehat{R_2}(\theta)$ and $\widehat{R_3}(\theta)$ are

$$\widehat{R_1}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{i\theta} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{pmatrix},$$

$$\widehat{R_2}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e^{i\theta} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\theta} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{pmatrix},$$

$$\widehat{R_3}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\theta} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{pmatrix}.$$

3.2.3 Controlled Not Gate (CNot)

Another kind of quantum gates are the controlled ones. This kind of gates involves two qubits where one qubit controls wheter the other is modified by the gate or not. This works in the following way: When the controlling qubit state value is $|1\rangle$, the quantum gate on the controlled qubit is activated, transfoming the state of the this qubit according to the kind of gate that we are using with the control operation. However, when the value of the controlling qubit is $|0\rangle$, the controlled qubit does not suffer any change and reamain at its previous state. The tranformatin matrix of this gate, where the controlling qubit is the number 1 and the controlled is the number two is [10]

$$\widehat{C}_U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{1,1} & U_{1,2} \\ 0 & 0 & U_{2,1} & U_{2,2} \end{pmatrix}.$$

where U is the transformation matrix of the gate that we are pairing with the control operation.

A very typical controlled quantum gate is the CNot. This gate flips the state of the controlled qubit if and only if the controllling qubit is on the state $|1\rangle$. Its circuit is

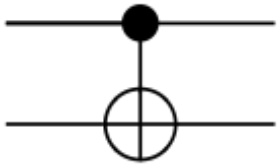


Figure 5: CNot gate [9]

and its matrix[13]

$$\widehat{C}_{Not[1,2]} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

When we move forward to systems with more than two qubits we increase the number of gates of this kind that we can build. For examples, in a three qubit system we can control the first qubit with the third,

$$\widehat{C}_{Not[3,1]} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix},$$

and viceversa

$$\widehat{C}_{Not[1,3]} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix},$$

or the third with the second

$$\widehat{C}_{Not[2,3]} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix},$$

among others.

3.3 Measuring final states

In a classical computer extracting the information from the bits after a computation is not even considered a problem. It is trivial. Also, reading the bits does not alter them.

However in quantum computation this is no longer the case. In fact, some of the biggest technological challenges in quantum computing are related to the fact that measuring on a quantum system usually produces its irreversible collapse.

It is important to remember at this point that quantum gates act as (reversible) unitary operators.

Before being measured the state of the system can be of two types: a basis state associated with an appropriate observable (the z components of the spins representing the qubits), or a superposition of these basis states.

When we perform the measurement we "collapse" the state into one of the possible outcomes (those with non-zero coefficients). The probabilities of the possible outcomes are are given by the modulus squared of the corresponding amplitudes[14].

4 Introduction to the Simulation of Quantum Algorithms

In order to perform the simulations we are going to use the program Wolfram Mathematica®. Using matrices we are going to emulate how the quantum states evolve through the quantum circuits and the operation of the quantum gates.

4.1 Generation of Quantum States

The first step in the simulation of a quantum computation is to generate the initial state for the system of qubits. In order to avoid the manual introduction of long lists of zeros and ones, we use a short program. It has two variables, one of them is the number of qubits. For the other we have taken advantage of the similarity between the way we represent the basis quantum states and binary code. For instance for the case of four qubits we will write

$$1 = |0000\rangle, 2 = 0001\rangle, 3 = |0010\rangle, 4 = |0011\rangle,$$
$$5 = |0100\rangle, 6 = |0101\rangle, 7 = |0110\rangle, 8 = |0111\rangle,$$
$$9 = |1000\rangle, 10 = |1001\rangle, 11 = |1010\rangle, 12 = |1011\rangle,$$
$$13 = |1100\rangle, 14 = |1101\rangle, 15 = |1110\rangle, 16 = |1111\rangle$$

These states are written as tensors products of $|0\rangle$ and $|1\rangle$. For instance

$$|1010\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \quad (110)$$

The tensor product operator in Mathematica can be easily implemented by using the Kronecker product. For this task we define a function called InitialStateGenerator. The whole program looks like this:

```
InitialStateGenerator[n_, numqubits_] := Transpose
                                          transposición
[Apply[KroneckerProduct, ReplaceAll[Flatten[{{Table
 aplica  producto Kronecker  sustituye todos  aplana  tabla
      [0, {k, 1, numqubits - Length[IntegerDigits[n - 1, 2]]}]},
                            longitud  dígitos de entero
    {IntegerDigits[n - 1, 2]}}, 2], {0 → {{1, 0}}, 1 → {{0, 1}}}]]]
     dígitos de entero
```

Figure 6: Initial State Generator

For the benefit of the reader we will briefly describe the Mathematica commands that we use in the implementation of quantum algorithms. For each simulation we will only describe the new commands use in order to avoid unnecessary repetitions.

For this function we have used:

- Transpose: It computes the transpose of a matrix or a vector.
- Table: It creates a table.

- Length: It express the length of a list.
- IntegerDigits: It creates integer digits in a certain basis.
- Flatten: It deletes inner brackets from a list.
- ReplaceAll: It replaces some specified element of the list by another.
- KroneckerProduct: It takes the Kronecker product of the elements specified inside the brackets.
- Apply: Performs a given operation to all the elements of a list.

First we have created a number in binary with Integerdigits. In order to be consistent we have to take the number minus one. So we have the following correspondence: $1 \to 0$, $2 \to 1$, $3 \to 10$, $4 \to 11$, $5 \to 100$, $6 \to 101$, $7 \to 110$, $8 \to 111$. However the problem that we find is that for the four first numbers, the program does not provide the first zeros. To correct that we have used a combination of the tools Table, Length, and IntegerDigits to introduce the missing zeros. Then, to take out the unwanted brackets we have used the Flatten command. Finally to build the vector we have replaced the $|0\rangle$ and the $|1\rangle$ according to

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} . \tag{111}$$

Then we compute the tensor product with the command KroneckerProduct. We do it with all the elements defined before. For this task we use the command Apply. Finally we use Transpose to transform the result from a line vector to a column one.

4.2 Measurement Program

The function of this program is to simulate the act of measuring the system of qubits. In real life, with real qubits this task is very difficult as explained before.

For this program we have defined three functions: PureStates, Measure and FinalCounter.

The purpose of PureStates is to create the possible final states. Measure simulates the act of measuring and FinalCounter performs a number of experiments and counts their outcomes.

This program can measure all kind of states, but at the end the result is going to be a basis state. Depending on the state that we collapse we are going to have a probabilistic distribution of basis states. In order to see that we have to carry on many experiments.

4.2.1 PureStates Function

The PureState function gives all the pure states in vector form for a given number of qubits. It has one variable, numqubits representing the number of qubits of the system that we are going to work with.

```
PureStates[numqubits_] := Table[n → StringJoin[StringJoin
                          tabla   une cadenas···  une cadenas de caracteres
    [Table["0", {k, 1, numqubits - Length[IntegerDigits[n - 1, 2]]}]],
     tabla                       longitud  dígitos de entero
          IntegerString[n - 1, 2]], {n, 1, 2^numqubits}]
          número entero como cadena de caracteres
```

Figure 7: PureStates

In order to define this function we have used several Mathematica commands, there are only two new ones with respect to the previous simulation, StringJoin whose purpose is to join several items into one single string and IntegerString whose purpose is the same as IntegerDigit but the number created is in a form of strings. The other ones used and described before are Table, Length, IntegerDigits, and IntegerDigit.

PureStates works in a similar way to the first part of InitialStateGenerator. There are some differences: here we are using strings so we have to join them in a single string with the command StrinJoin and that we include the non binary number plus one besides the state. Also, all this happens inside Table. The size of the table depends on the number of qubits according to $2^{No of qubits}$. For example, for four qubits, this function is going to be:

```
PureStates[4]

{1 → 0000, 2 → 0001, 3 → 0010, 4 → 0011, 5 → 0100, 6 → 0101, 7 → 0110, 8 → 0111,
 9 → 1000, 10 → 1001, 11 → 1010, 12 → 1011, 13 → 1100, 14 → 1101, 15 → 1110, 16 → 1111}
```

Figure 8: Pure States for four qubits

4.2.2 Measure Function

This function simulates the act of measuring. For this we use a random number generator. The function variables are ψ that is the quantum state that we want to measure and the number of qubits.

```
Measure[ψ_, numqubits_] := Module[{r = RandomReal[WorkingPrecision -> 30]},
                           módulo   real aleatorio  precisión operativa
  Min[Flatten[Position[Table[Boole[(r - Sum[Map[Abs[#]^2 &, ψ][[k]], {k, 1, l}])[[1]] < 0],
  mí···  aplana  posición  tabla  función car···  s···  apl···  valor absoluto
      {l, 1, 2^numqubits}], 1]]] /. PureStates[numqubits]]
```

Figure 9: Measure Function

For the definition of this function we have used some new Mathematica commands:

- Module: This defines an structure that allows you to work with local variables.
- RandomReal: It generates a random real number between 0 and 1. The command inside it, WorkingPrecision is to adjust the number of decimals.
- Min: Selects the minimum value among various values.

- Position: Indicates the position of the elements of a table.
- Boole: Returns 1 if the expression is True and 0 if it is False.
- Sum: Sums all the elements inside it.
- Map: It applies to each element on the expression the first part of the brackets.

Also we use Table and Flatten.

Inside the Module we have two separated processes. The purpose of the first one is to generate a random number with RandomReal in order to introduce the randomness characteristic of the quantum world. $r =$ RandomReal[WorkingPrecision$-> 30$]

The other one is to set up a "system of boxes" labeled by the basis states. The size of the boxes is given by the squared module of amplitude of each basis so the size is equal to the probability of being measured. Is important to notice that all the basis states with amplitudes equal to zero disappear in this step. Then we find out "in which of the boxes the random number falls". This gives us the state that remains after the measurement with a probability controlled by the amplitudes used to write the state in the fundamental basis.

4.2.3 FinalCounter Function

The purpose of this function is to repeat the experiment as many times as we want to get representative values. It has two variables: numqubits to introduce how many qubits are we using and p, the number of experiments.

```
FinalCounter[p_, numqubits_] := Counts[Table[Measure[ψ, numqubits], {k, 1, p}]]
                                  conteos  tabla
```

Figure 10: FinalCounter

The only new Mathematica commands that we are using is Counts. This command counts how many times an element of a list appears. The list that we are using is a table that contains the results of p experiments performed on the state ψ. If p is large enough we get significant information about the quantum state before the collapse, the main goal of measuring.

4.3 Implementation of the Hadamard Gates

As explained before one of the most common quantum gates is the Hadamard. For this reason we have define an independent function to compute it. In order to do this we just need

$$\widehat{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} . \tag{112}$$

and find the way to replicate the process detailed in section 3.2.1.

This function is called Hadamard. Its arguments are the number of qubits involved (numqubits) and the position of qubit that it acts upon (particula). The

main ingredient is the KroneckerProduct mathematica command which computes the tensor products of the identity matrices and the $\widehat{H}$.

```
H = {{1, 1}, {1, -1}} / Sqrt[2]
                        raíz cuadr

Hadamard[numqubits_, particula_] :=
 KroneckerProduct[ IdentityMatrix[2^(particula - 1)], H,
 producto Kronecker   matriz identidad
  IdentityMatrix[2^(numqubits - particula)]]
  matriz identidad
```

Figure 11: Hadamard function

5 Grover Algorithm

The Grover algorithm is a tool to search a unique item in an unordered list of N elements. The only way a classical computer algorithm can solve this problem is starting from the beginning, until the unique item is found. With a bit of luck, the element might be at the beginning of the list, however that is not the most common case. On average, a classical computer has to try on at least around half of the elements of the database: $T = \frac{N+1}{2}$, therefore, the time needed to solve the problem will be around T times the needed time to check one of them. The Grover algorithm has the potential to reduce the same search to $\sqrt{T}$ [15], a quadratic speed up. For small databases the improvement might not be huge, but when they get bigger the difference becomes relevant. The Grover algorithm could solve, if quantum computers improve (mostly by getting bigger so they have more qubits), some search problems that before were beyond the reach of any algorithm in a classical computer.

The improvement in search problems provided by the Grover algorithm relies on the quantum phenomena know as superposition. The number of qubits involved in the calculation are heavily related with the size of the database (N). The first step, like in many others quantum algorithms, is to transform the initial state into an equal superposition among all the qubits used (n). An equal superposition state, as commented before, is a state where all the basis have the same amplitudes, in other words, the same likelihood of being found if a measurement is carried on.

The next step is to transform this superposition state into the solution of the searching problem. This is done with the main body of the algorithm. It is composed by the Oracle and the Diffusion operator. You have to repeat this part a precise number of times (later we will see how many). Once the algorithm is run the required number of times we get a final state where one of the amplitudes is large while all the others are very small. This means that the searched object (unique element) is located at the position corresponding to this large amplitude. As the difference among the coefficients is huge, the majority of times you measure the final state, results on the right answer. To complete this brief explanation and to help with the further understanding of the algorithm, is important to visualize its quantum circuit.

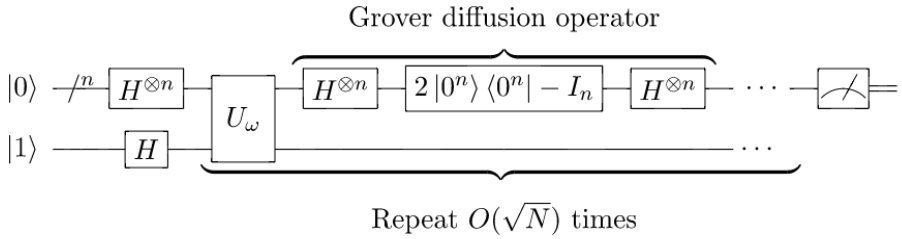


Figure 12: Grover Circuit [11]

In the figure we can see two main lines. The upper one refers to the n qubits involved in the search. You can recognize on its way the previously mentioned quantum gates, the Hadamards ($H^{\otimes n}$), the Oracle (U_w) and the ones inside the diffusion operator. We have to repeat the loop formed by the Oracle and the

diffusion operator certain number of times. The other line is an extra qubit, called the scratch qubit, whose task is helping to avoid the decoherence of the system.

5.1 How it works

In this section we are going to explain the algorithm in detail. The main goal is to provide to the reader with all the necessary tools to fully understand it. In order to do so, we are going to rely on the theoretical basis that we have explained so far. The description is going to be composed of two interconnected parts: a general explanation (valid for any number of qubits) and a parallel example of three qubits. Each time we explain something in general explanation we will show how it works in this concrete example. With this we hope that the discussion becomes more understandable.

The needed number of qubits for the Grover algorithm, besides the scratch qubit, is n, being $n = \log_2 N$. This result comes from $N = 2^n$ where N is the number of elements of the database. The combined initial state of the n qubits has to be $\psi = |0, \ldots, 0\rangle$ (the number of elements inside the kets is equal to n, one for each qubit). To save time and space we are going to write this initial state in a more compact way $\psi = |0\rangle^{\otimes n}$. This is a vector of 2^n elements and it is calculated from the individual qubit states with the tensor product. This has been already explained in detail in 3.1.2. For our small example the number of qubits is three $n = 3$ so the database size is $N = 2^3 = 8$. Analogously the basis to write the states is composed by 8 vectors ($|000\rangle, |001\rangle, ...|111\rangle$). Following the things said before, the qubits initial state has to be $\psi = |000\rangle$.

The first step is to achieve an equal superposition. We apply the Hadamard gates. It was described in detail in 11. The outcome of the gate provides a state where all the coefficients of the basis have the same probability of being measured. Is important to remember that in order to obtain a full superposition of all the qubits we have to apply all the corresponding adjusted Hadamard gates (one for each qubit). Otherwise the superposition would be only among the qubits that pass through them. When done correctly the initial state is transformed into:

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \,, \tag{113}$$

where we denote

$$H^{\otimes n} = \widehat{H}_1 \widehat{H}_2 \ldots \widehat{H}_n \tag{114}$$

and

$$\widehat{H}_1 = \widehat{H} \otimes \mathbb{I} \otimes \ldots \otimes \mathbb{I} \tag{115}$$

$$\widehat{H}_2 = \mathbb{I} \otimes \widehat{H} \otimes \mathbb{I} \otimes \ldots \otimes \mathbb{I} \tag{116}$$

For our example, we have to use three 8×8 Hadamards. We do not have to calculate them because they were computed before in 103. The calculation looks like this:

$$|\psi\rangle = H^{\otimes 3}|0\rangle^{\otimes 3} = \frac{1}{2\sqrt{2}}|000\rangle + \frac{1}{2\sqrt{2}}|001\rangle + ... + \frac{1}{2\sqrt{2}}|111\rangle = \frac{1}{2\sqrt{2}} \sum_{x=0}^{7} |x\rangle \tag{117}$$

All the coefficients of this state have the same magnitude. We show this in the following figure 13.

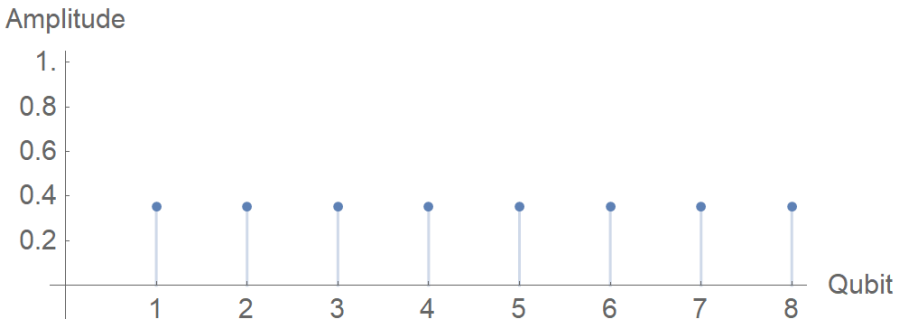


Figure 13: Amplitudes after the actions of the Hadamard gates

Once we have the qubits in the desired state we reach the main body of the Grover algorithm. This set of operations has to be repeated a precise number of times in order to successfully complete the search. This part uses three different quantum gates. The oracle, the Hadamards and a conditional phase shift. The precise number of times that we have to repeat the process in order to be optimal is the integer approximation to

$$\frac{\pi}{4}\sqrt{2^n}. \tag{118}$$

For our example it is $\frac{\pi}{4}\sqrt{2^3}$ which is about 2.22 so we have to repeat the central process two times.

As shown in the circuit the first gate of the loop is the Oracle. Here is where the information that we are looking for is encoded. Its task is to rotate by π radians the phase of the basis coefficient (the one corresponding to the location of the searched element) while leaving all the others untouched. Unfortunately this phase shift is impossible to detect when measuring. You have to keep in mind that the probabilities of measuring as outcome state each of the options are the squares of the modules of the amplitudes and therefore the signs "disappear" when we take only into account the modulus. The task of the other part of the loop is to transform the phase shift into an amplitude change.

The oracle transformation can be numerically expressed like [12]

$$x = (-1)^{f(x)}x, \tag{119}$$

where $f(x) = 0$ when x is not the element that we are searching and $f(x) = 1$ when x is the element that we are searching.

Another way to express this is with a transformation matrix. The matrix will look like the identity matrix $\mathbb{I}$ but it will have a minus one in the location of the diagonal corresponding to the searched element [10].

For our example we are going to suppose that the searched element is in the

second position. The transformation matrix will look like:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{120}$$

The Oracle gate transforms our equal superposition ψ into a new state where the second element has been shifted π radians. The new state is,

$$|\psi\rangle = \frac{1}{2\sqrt{2}}|000\rangle - \frac{1}{2\sqrt{2}}|001\rangle + ... + \frac{1}{2\sqrt{2}}|111\rangle = \frac{1}{2\sqrt{2}}\sum_{x=0}^{7}|x\rangle. \tag{121}$$

and graphically:

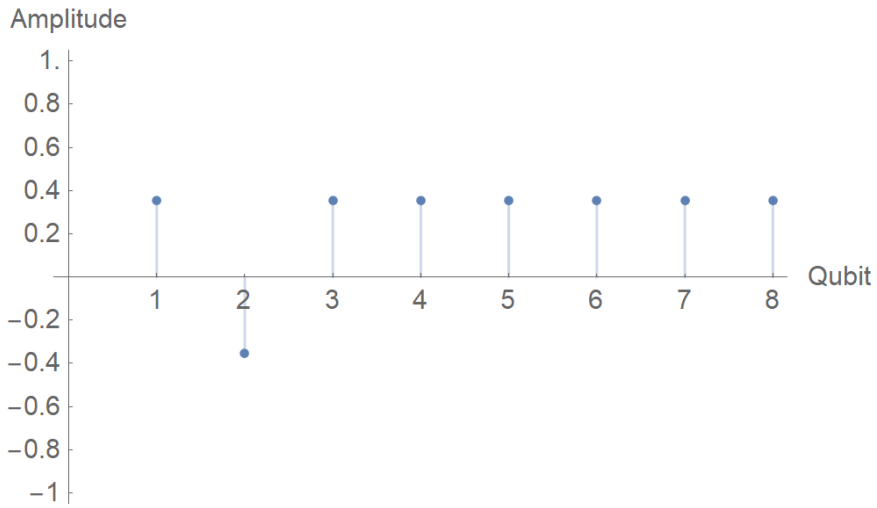


Figure 14: Amplitudes after the first action of the Oracle gate

where the change of sign can be seen.

As previously mentioned, this transformation is not enough. We need more gates to transform the phase shift into a measurable variation. The measurable variation in order to be effective needs to have a very important feature: it has to affect the modulus and it has to be large (otherwise we will obtain many times a wrong solution when measuring). This part is called the diffusion transformation. Its basic principle of operation is to exploit the difference on the amplitude of the searched state (reversed by the phase shift) with respect the average amplitude of all the possible states.

Getting back to our example, we can see this easily. The amplitude of the states -taking out the $|001\rangle$- is $\frac{1}{2\sqrt{2}}$ while the amplitude of the special state is $-\frac{1}{2\sqrt{2}}$. If

we calculate the average (μ) we get

$$\mu = \frac{1}{8}\left((8-1)\frac{1}{2\sqrt{2}} - \frac{1}{2\sqrt{2}}\right) = \frac{3}{8\sqrt{2}}\,. \tag{122}$$

We can observe that the difference between the average and the normal states is much smaller (in fact if N is sufficiently large it will be negligible)

$$\frac{1}{2\sqrt{2}} - \mu = \frac{1}{2\sqrt{2}} - \frac{3}{8\sqrt{2}} = \frac{1}{8\sqrt{2}} = 0.09, \tag{123}$$

than the difference between the average and the special case,

$$|-\frac{1}{2\sqrt{2}} - \mu| = \frac{1}{2\sqrt{2}} + \frac{3}{8\sqrt{2}} = \frac{7}{8\sqrt{2}} = 0.62. \tag{124}$$

Taking advantage of this, the transformation carried out by the diffusion operator can be expressed like [16]

$$\sum_{x=0}^{n} a_x|x\rangle \rightarrow \sum_{x=0}^{n}(2\mu - a_x)|x\rangle. \tag{125}$$

Each time we pass through the loop we increase the difference in the amplitudes until we reach the limit set by $\frac{\pi}{4}\sqrt{2^n}$ where it starts to decrease. In fact it is cyclical. We will check this with the Mathematica simulation.

We need now some quantum gates. They are the usual Hadamard gate, denoted in the figure by $H^{\otimes}n$ but also a new one, called J and denoted in the figure with its operation, $2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - \mathbb{I}$ [12].

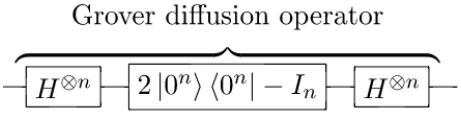


Figure 15: Diffusion Operator circuit [11]

The task of the J is to apply a conditional phase shift of -1 to all the states but $|0\rangle$. It is represented by the unitary operator $2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - \mathbb{I}$. In the case $n = 1$ (just one qubit) it is

$$J_1 = 2|0\rangle\langle 0| - \mathbb{I} = 2\begin{pmatrix}1\\0\end{pmatrix}\otimes(1\ 0) - \mathbb{I} = 2\begin{pmatrix}1&0\\0&0\end{pmatrix} - \begin{pmatrix}1&0\\0&1\end{pmatrix} = \begin{pmatrix}1&0\\0&-1\end{pmatrix}. \tag{126}$$

For our three-qubit system the result is

$$J_3 = 2|0\rangle^{\otimes 3}\langle 0|^{\otimes 3} - \mathbb{I}_3 = \begin{pmatrix}1&0&0&0&0&0&0&0\\0&-1&0&0&0&0&0&0\\0&0&-1&0&0&0&0&0\\0&0&0&-1&0&0&0&0\\0&0&0&0&-1&0&0&0\\0&0&0&0&0&-1&0&0\\0&0&0&0&0&0&-1&0\\0&0&0&0&0&0&0&-1\end{pmatrix}. \tag{127}$$

Now we combine it with the Hadamards, as shown in the figure15:

$$D = H^{\otimes n}(2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - \mathbb{I})H^{\otimes n}. \tag{128}$$

To expand this expression we introduce the following notation:

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle =: |H\rangle^{\otimes n} \tag{129}$$

Now we can easily compute 128.

$$\begin{aligned} D =& H^{\otimes n}(2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - \mathbb{I})H^{\otimes n} = 2H^{\otimes n}|0\rangle^{\otimes n}\langle 0|^{\otimes n}H^{\otimes n} + H^{\otimes n}H^{\otimes n} \\ =& 2|H\rangle^{\otimes n}\langle H|^{\otimes n} - \mathbb{I}_n \end{aligned}$$

To show how this procedure works in our example, we have to remember that $\langle H|^{\otimes 3}|H\rangle^{\otimes 3} = 1$ and that $\langle H|^{\otimes 3}|001\rangle = \frac{1}{2\sqrt{2}}$.

$$\psi = (2|H\rangle^{\otimes 3}\langle H|^{\otimes 3} - \mathbb{I}_3)|\psi\rangle \tag{130}$$

$$= (2|H\rangle^{\otimes 3}\langle H|^{\otimes 3} - \mathbb{I}_3)(|H\rangle^{\otimes 3} - \frac{2}{2\sqrt{2}}|001\rangle \tag{131}$$

$$= 2|H\rangle^{\otimes 3}\langle H|^{\otimes 3}|H\rangle^{\otimes 3} - |H\rangle^{\otimes 3} - \frac{2}{\sqrt{2}}|H\rangle^{\otimes 3}\langle H|^{\otimes 3}|001\rangle + \frac{1}{\sqrt{2}}|001\rangle \tag{132}$$

$$= 2|H\rangle^{\otimes 3} - |H\rangle^{\otimes 3} - \frac{-2}{\sqrt{2}}(\frac{1}{2\sqrt{2}})|H\rangle^{\otimes 3} + \frac{1}{\sqrt{2}}|001\rangle \tag{133}$$

$$= |H\rangle^{\otimes 3} - \frac{1}{2}|H\rangle^{\otimes 3} + \frac{1}{\sqrt{2}}|001\rangle \tag{134}$$

$$= \frac{1}{2}|H\rangle^{\otimes 3} + \frac{1}{\sqrt{2}}|001\rangle \tag{135}$$

$$= \frac{1}{2}\left[\frac{1}{2\sqrt{2}}\sum_{x=0}^{7}|x\rangle\right] + \frac{1}{\sqrt{2}}|001\rangle \tag{136}$$

$$= \frac{1}{4\sqrt{2}}\sum_{x=0(x\neq 3)}^{7} + \frac{1}{4\sqrt{2}}|001\rangle + \frac{1}{\sqrt{2}}|001\rangle \tag{137}$$

$$= \frac{1}{4\sqrt{2}}\sum_{x=0(x\neq 3)}^{7} + \frac{5}{4\sqrt{2}} \tag{138}$$

Summarizing, the computation leaves the amplitudes in $\frac{1}{4\sqrt{2}}$ for the normal states and in $\frac{5}{4\sqrt{2}}$ for the searched one.

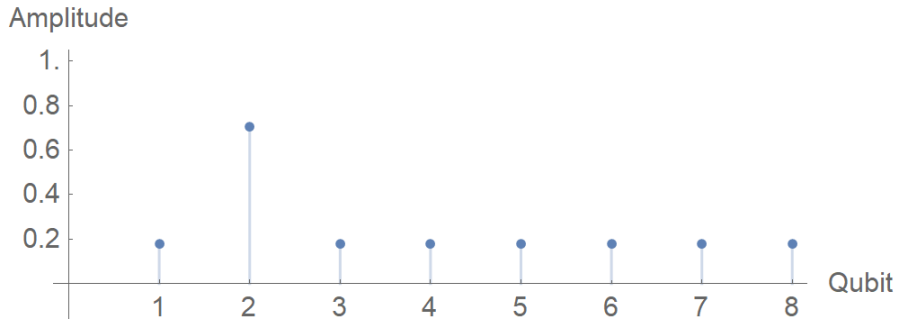


Figure 16: Amplitudes after the first time through the loop

Then you have to repeat the loop the indicated number of times 118, in our case two times. Through a similar computation you obtain that the amplitudes have changed in the following way: the magnitude corresponding to the searched entry goes up to a value of $\frac{11}{8\sqrt{2}}$ whereas the remaining ones became $\frac{1}{8\sqrt{2}}$ (which are 11 times smaller).

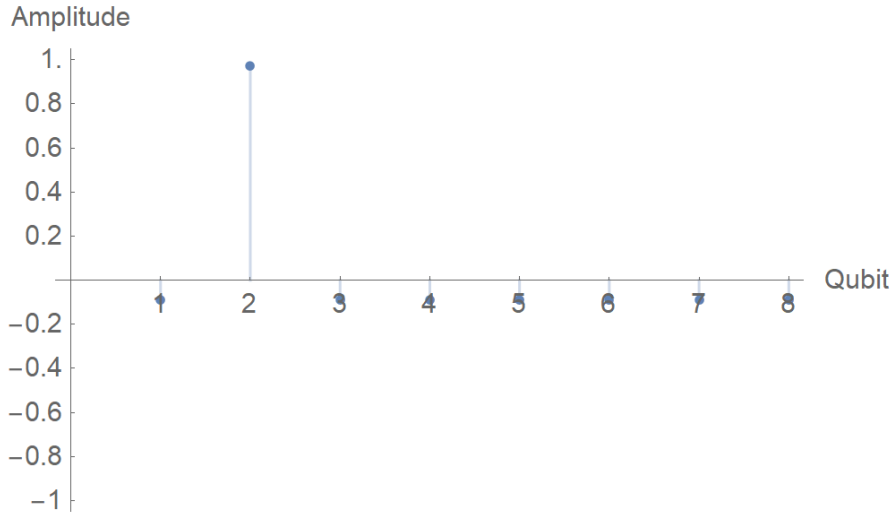


Figure 17: Amplitudes after the second time through the loop

These amplitudes lead to the following probabilities when measuring:

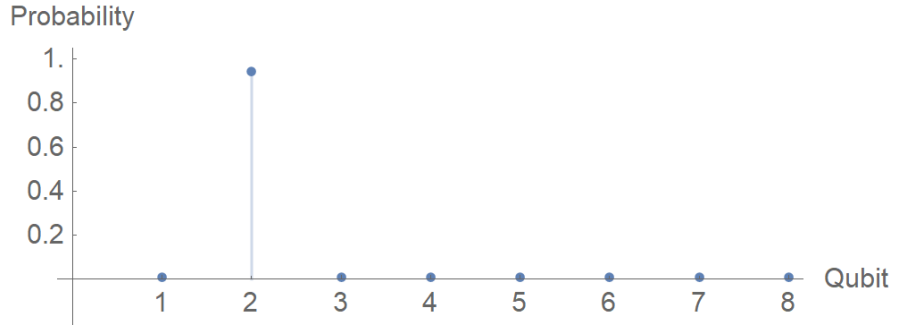


Figure 18: Probabilities after the implementation of the Grover algorithm

We can see easily that the difference is huge. If we compute the sum of probability of all the non-searched elements we obtain $\frac{7}{128}$ while the probability of the searched one is $\frac{121}{128}$. This proves that the efficiency and accuracy of the Grover algorithm is remarkable.

5.2 Mathematica Implementation of the Grover algorithm

Once we know how the algorithm works, it is time to perform the simulations in Mathematica in order to test it. To build the Grover program, we are going to use a similar procedure to the one used with the previous Mathematica programs (InitialStateGenerator, Measure and Hadamard). To help the computer with the simulation, we are going to use sparse matrices. They are useful when you are using big matrices with many zeros, because they do not store every element. This reduces the memory usage and enhances the computation speed.

In order to implement the Grover algorithm in Mathematica we need to define the gates representing the Oracle and the diffusion operator. Then, we are going to put together the Oracle and the diffusion operator in another function, the LoopGrover. Finally we will combine all these elements in an single function, EstadoFinal, to compute the final state. To measure the resulting state, we are going to use the measure programm described before.

5.2.1 Oracle

To compute its transformation matrix we have define a function called Oraculo. It has two variables, numqubits, (the number of qubits) and particle which provides the position in the database of the searched element.

For the computation we have used the similarity between the identity and the Oracle matrix. We use the Mathematica command IdentityMatrix to create an identity matrix proportional to the number of qubits: $2^{\text{numqubits}}$. Then we use ReplacePart to change one 1 from the diagonal of the identity matrix by a -1. The coordinates of this replacement are provided by the variable particle.

```
Oraculo[numqubits_, particle_] :=
  SparseArray[ReplacePart[IdentityMatrix[2^numqubits],
  array disperso  sustituye una ···  matriz identidad
    {particle, particle} → -1]];
```

Figure 19: Oracle function

5.2.2 Diffusion Operator

The diffusion operator is more complex than the Oracle because it involves more than one quantum gate. These are the J and the Hadamards.

The J is a new gate for us. Therefore we have to create a new function to compute it. We have used a procedure similar to the one followed in the Oracle. The transformation matrix associated with this gate is like the minus identity matrix but with a 1 instead of a -1 in the first position. So using the same

Mathematica commands as in the Oracle, we create a new function called J. It has just one variable, the number of qubits because unlike the Oracle the 1 is always in the same position so we do not need to specify where it is each time we use the gate.

```
J[numqubits_] := -ReplacePart[IdentityMatrix[2^numqubits], {1, 1} → -1];
                  sustituye una ···  matriz identidad
```

Figure 20: J function

The other part of the diffusion operator involves the Hadamards. We just have to multiply the Hadamards corresponding to each qubit, introduce then the J matrix and multiply the result by all the Hadamards again. We do this with the help of the function ProductoHadamards. This function depends on the number of qubits.

```
ProductoHadamards[numqubits_] :=
  Apply[Dot, Table[Hadamard[numqubits, k], {k, 1, numqubits}]];
  aplica  pr···  tabla
```

Figure 21: Hadamard products

The Grover diffusion operator for numqubits is implemented as the function GroverDiffusionOperator.

```
GroverDiffusionOperator[numqubits_] :=
 ProductoHadamards[numqubits].J[numqubits].ProductoHadamards[numqubits]
```

Figure 22: Grover Diffusion Operator

5.2.3 The loop

With the Oracle and the diffusion operator we can define the loop, which is the main body of the algorithm body. To achieve an optimal operation it has to be repeated a certain number of times given by the integer approximation to $\frac{\pi}{2}\sqrt{2^{number\ of\ qubits}}$.

To compute it in Mathematica we have created a function called LoopGrover. It depends on the number of qubits and on where the searched object is because it contains the Oracle.

The number of times that the loop needs to be repeated is computed by the loop itself. For this task we have written the expression in Mathematica, $\frac{\pi}{2}\sqrt{2^{number\ of\ qubits}}$, and used the Round command to convert its decimal result into an integer.

We have to multiply first the Grover diffusion operator with the Oracle and then proceed with the repetitions. The best way to implement this is by using the command MatrixPower to multiply the result of the previous operation by itself as many times as needed.

```
LoopGrover[numqubits_, particle_] :=
 MatrixPower[GroverDiffusionOperator[numqubits].Oraculo[numqubits, particle],
 potencia matricial
  Round[(π / 4) Sqrt[2^numqubits]]]
  entero más pró··· raíz cuadrada
```

Figure 23: Grover Loop

5.2.4 The EstadoFinal function

Now we have all the parts needed to perform a full simulation of the Grover algorithm: the initial state generator, the Hadamards and the loop. We just have to put them together. This is done with another function called EstadoFinal with variables equal to the number of qubits, the position of the searched object is and the initial state.

```
Estadofinal[numqubits_, particle_, n_] :=
 LoopGrover[numqubits, particle].ProductoHadamards[numqubits].
  InitialStateGenerator[n, numqubits]
```

Figure 24: Grover final state

This function gives us the final quantum state after the operation of the Grover algorithm. We use the measure program to extract the information from this state.

5.3 The simulations

We test our Mathematica implementation of the Grover algorithm by performing several simulations, first with three qubits and then with seven. We will use the measure program to extract the information from the final state.

We will also see the effect of the repetitions of the loop on the final state and the efficiency of the algorithm. Finally, we will show how the algorithm works if the Oracle contains more than one non zero entries.

For the simulation we are going to use several Mathematica commands to plot the different results to make then easier to understand. In particular we use ListPlot to plot the final states amplitudes and probabilities.

5.3.1 Grover with three qubits in Mathematica

We first check the result for three qubits (numqubits=3) and a database (implemented as the Oracle) where the only non zero entry is the second (particle=2). The initial state is $|000\rangle$ (n=1). This is given by

```
Estadofinal[3, 2, 1]
```

$$\left\{\left\{-\frac{1}{8\sqrt{2}}\right\},\left\{\frac{11}{8\sqrt{2}}\right\},\left\{-\frac{1}{8\sqrt{2}}\right\},\left\{-\frac{1}{8\sqrt{2}}\right\},\left\{-\frac{1}{8\sqrt{2}}\right\},\left\{-\frac{1}{8\sqrt{2}}\right\},\left\{-\frac{1}{8\sqrt{2}}\right\},\left\{-\frac{1}{8\sqrt{2}}\right\}\right\}$$

Figure 25: Final state of the simulation for three qubits

As we can see we get the results mentioned in the text 5.1. The largest amplitude corresponds to the second entry. We also check what happens if we run the loop once. For this we modify the function Loop into Loop2.

```
LoopGrover2[numqubits_, particle_, reps_] :=
 MatrixPower[GroverDiffusionOperator[numqubits].Oraculo[numqubits, particle], reps]
 potencia matricial
```

Figure 26: Loop 2

The only difference is that here we can control the number on repetitions with the variable rep. Due to this additional variable we have to modify also EstadoFinal into EstadoFinal2 where inside we have Loop2 instead of Loop.

```
Estadofinal2[numqubits_, particle_, n_, reps_] :=
 LoopGrover2[numqubits, particle, reps].ProductoHadamards[numqubits].
  InitialStateGenerator[n, numqubits]
```

Figure 27: Grover final state 2

Now we can check that our program also works for the intermediate steps.

```
Estadofinal2[3, 2, 1, 1]
```

$$\left\{\left\{\frac{1}{4\sqrt{2}}\right\},\left\{\frac{5}{4\sqrt{2}}\right\},\left\{\frac{1}{4\sqrt{2}}\right\},\left\{\frac{1}{4\sqrt{2}}\right\},\left\{\frac{1}{4\sqrt{2}}\right\},\left\{\frac{1}{4\sqrt{2}}\right\},\left\{\frac{1}{4\sqrt{2}}\right\},\left\{\frac{1}{4\sqrt{2}}\right\}\right\}$$

Figure 28: Simulation with three qubits after one repetition

5.3.2 Grover with seven qubits

Once that we are sure that the program produces the same outputs that we compute by hand, we can go after more ambitious targets than just three qubits. The program runs for any number of qubits, however our computers cannot handle more than a few tens of qubits even using sparse matrices. Therefore we are going to run a 7 qubits system. It is large enough to see interesting effects but small enough to work in a reasonable time with our limited computational resources.

In this case we have numqubits=7, the non zero entry of our database is the tenth (particle=10) and the initial state and is $|0000000\rangle$ so we put $n = 1$.

```
ψ = Estadofinal[7, 10, 1]
```

Figure 29: Grover final state

The following figure shows the resulting amplitudes. Notice that now the states have $128 = 2^7$ components. As we can see the largest amplitude correspond to the tenth component.

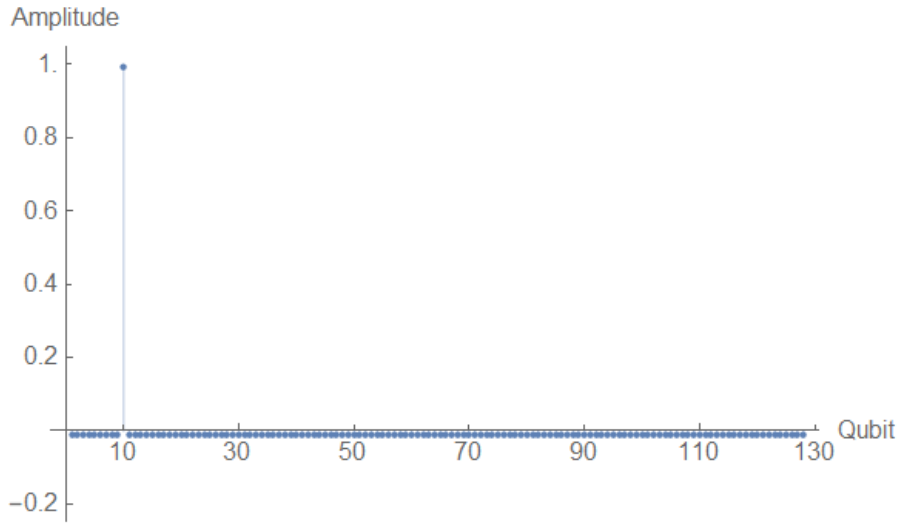


Figure 30: Amplitudes 7 qubits

The probabilities are the squares of the modulus of these amplitudes:

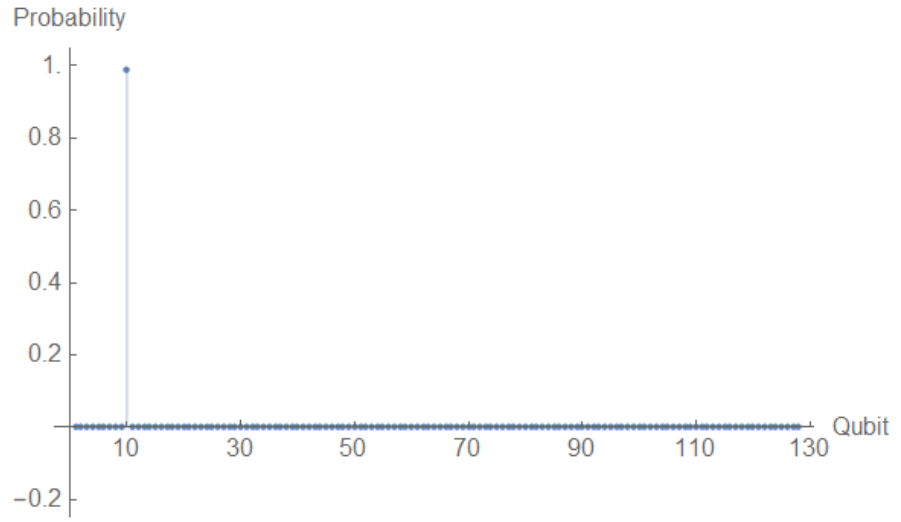


Figure 31: Probabilities for 7 qubits

To complete the analysis, we have used the measure program to simulate a number of measurement on the final states. A sample result is the following.

```
FinalCounter[100, 7]
⟨| 001001 → 99, 110110 → 1 |⟩
```

Figure 32: Measurements 7 qubits

Out of the 100 measurement we made, just one did not correspond to the right solution.

5.3.3 The Repetitions

Here we are going to see the effect of the number of loop repetitions on the amplitudes of the final state. We are going to compute, from final state, the amplitude corresponding to searched entry as a function of the number of the repetitions of the loop. We do this to check that the optimal number of repetitions is the closets integer to $\frac{\pi}{4}\sqrt{2^{\text{numqubits}}}$. For this simulations we have to use the modified version of the Grover used to compute the intermediate step with three qubits 5.3.1.

First we do this for the three qubit system. We compute the probability of the searched entry being found each time we pass through the loop. Then we plot it against the number of repetitions. The following plot shows the result.

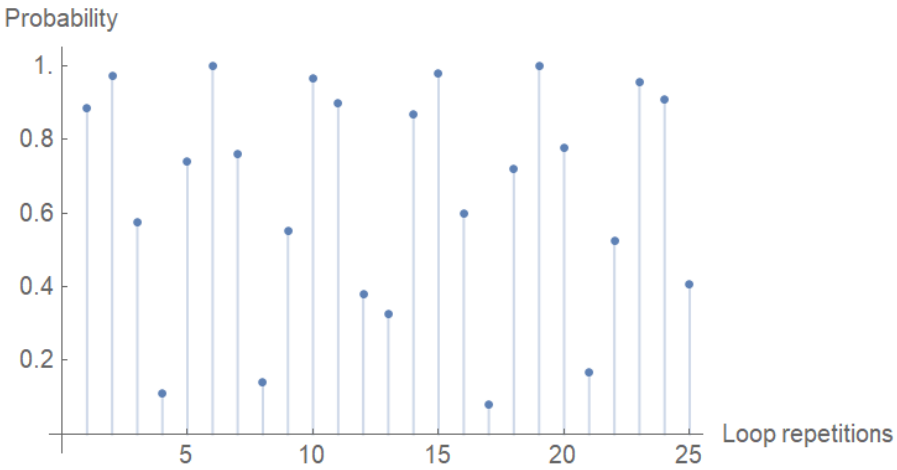


Figure 33: Probability in terms of the number of repetitions for 3 qubits

It is not easy to see a pattern here and one could think that the distribution is kind of random. However if we join the dots with the Mathematica command ListLinePlot this changes.

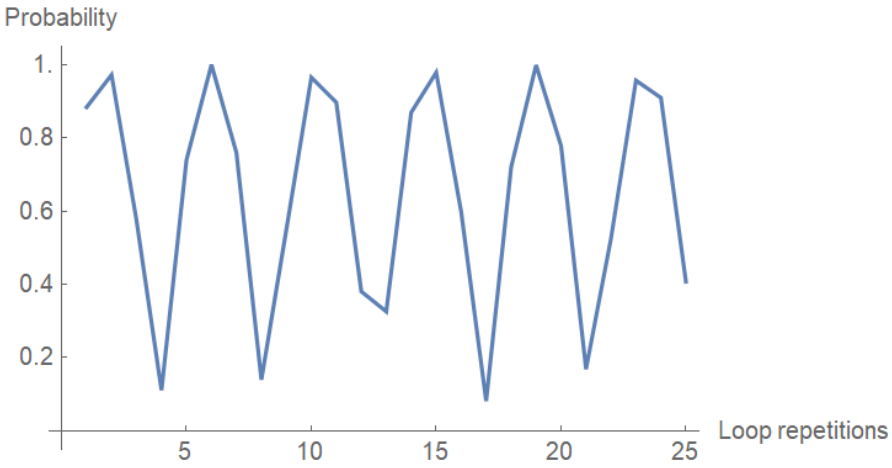


Figure 34: Probability in terms of the number of repetitions for 3 qubits

Here we see something resembling a periodic distribution with a period around four, and a first maximum occurring for two repetitions.

If we do the same for a 7 qubits system the periodicity becomes much more clear. The period is clearly the double of the number of loop repetitions and it peaks in the middle of each period.

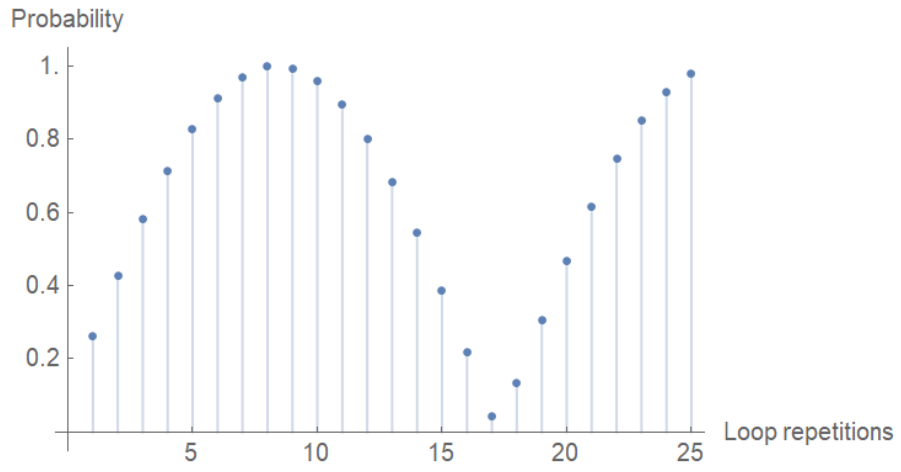


Figure 35: Grover final state

As we can see in these examples, $\frac{\pi}{4}\sqrt{2^{N^{O}\, of\ qubits}}$ is the optimal number of repetitions of the loop in the Grover algorithm.

5.3.4 Databases with several nonzero entries

With the Grover algorithm we can efficiently look for one single item in a database. But could we perform this kind of search to find more than one item at the same time? We are going to adapt the program in order to do so. The modification comes in the Oracle. Now we have to be able to input more minus ones in the matrix. We have to modify several functions to take this into account:

```
Oraculo3[numqubits_, particle_, particle1_] :=
  SparseArray[ReplacePart[IdentityMatrix[2^numqubits],
   {{particle, particle} → -1, {particle1, particle1} → -1}]];

LoopGrover3[numqubits_, particle_, particle1_] :=
 MatrixPower[GroverDiffusionOperator[numqubits].Oraculo3[numqubits, particle, particle1],
  Round[(π / 4) Sqrt[2^numqubits]]]

Estadofinal3[numqubits_, particle_, particle1_, n_] :=
 LoopGrover3[numqubits, particle, particle1].ProductoHadamards[numqubits].
  InitialStateGenerator[n, numqubits]
```

Figure 36: Grover for two items

An interesting question at this point is if the number of times that we have to run the loop changes or not. As we show in the following example it actually does. Picture 37 shows the magnitude of the probabilities after running the loop seven

times for a seven qubit system. Although the searched entries clearly stand out, it is actually possible to get a better result with fewer repetitions of the loop.

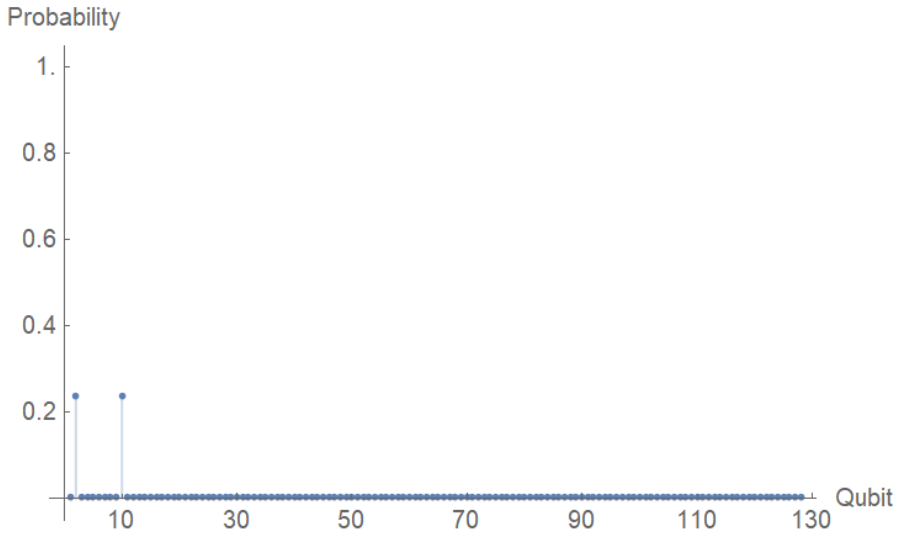


Figure 37: Grover for two items final state

If we measure this final state we would need many measurements to be sure of the results, and this is not useful in a real application of the algorithm.

To see how the Grover algorithm works in the present case we have to slightly modify the program in an obvious way. The results show that the number of optimal repetitions depends on the number of non zero entries in the database. This is shown in figures 38 and 39.

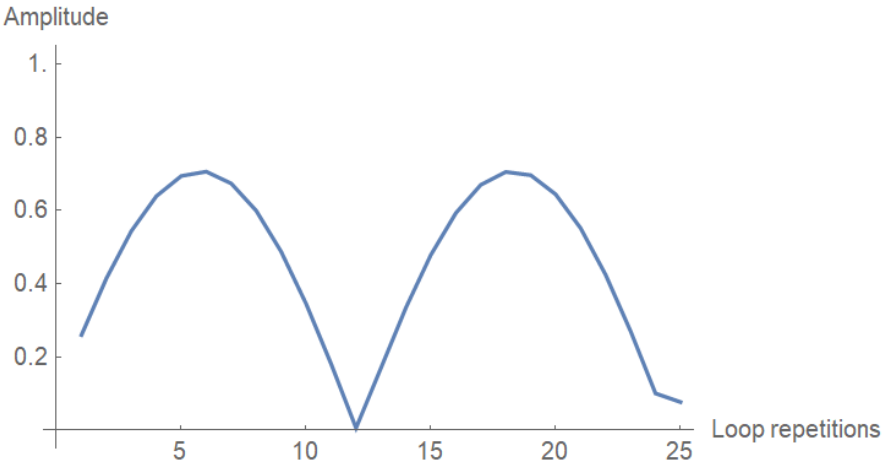


Figure 38: Looking for 2 elements with 7 qubits

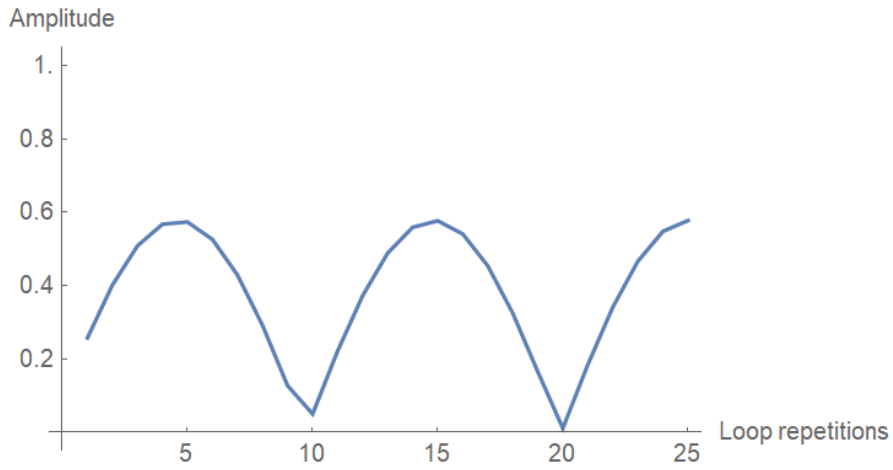


Figure 39: Looking for 3 elements with 7 qubits

We can see that the number of loop iterations decreases as the number of entries increases. In fact it can be shown that the optimal number of operations is given by [17]:

$$\frac{\pi}{4}\sqrt{\frac{2^{N^o\, of\ qubits}}{N^o\, of\ searched\, elements}}. \tag{139}$$

Our simulations confirm this result.

5.4 Conclusions and comments on the Grover algorithm

From our simulations and from all the papers I have read, my personal feeling is that the Grover algorithm could mean a significant improvement in the resolution of some complex tasks like the collision problem [18], the machine learning process [19], breaking passwords or the calculation of means and medians in statistical distributions.

The main concern with the Grover algorithm, besides the well known difficulties of the physical implementation of quantum computers, is the creation of the Oracle. As it is a quantum gate, we cannot plug the real database into the algorithm, instead we would to "translate" it to the Oracle. This procedure might, in fact, take longer than the time needed to run the algorithm.

From my point of view the best way to start learning how quantum algorithms work is by studying by this one. When Lov K. Grover published his paper back in 1996 [15] he proved for the first that quantum computers have the real potential of being better than the classical ones. Besides this fact, this algorithm can be understood in a very visual way. For me this was very helpful because when you understand how the procedure works its easier to dive into the maths and the physics behind it.

6 The Shor algorithm

The efficient factorization of large integer numbers in prime factors is a very difficult problem even for the best classical computers. However, this could change thanks to quantum computation and the Shor algorithm. Before this quantum algorithm appear, factorizing a number N required at least a sub-exponential time, $O(e^{1.9(logN)^{\frac{1}{3}}(loglogN)^{\frac{2}{3}}})$ [20]. This was achieved by using a general number field sieve algorithm. The Shor algorithm running in a quantum computer, could improve this to a polynomial time (i.e. is a polynomial of $log(N)$) [21]. This could change many things. One particular fields would suffer a major shift: cryptography. This algorithm could mean the end for some approaches to public key cryptography such as the RSA scheme. This is based on the general assumption that big numbers can be easily obtained through multiplication but it is almost impossible to factor them. If we overcome the difficulties of creating a practical quantum computer able to run the Shor algorithm this might not be anymore true and the RSA cryptography could become obsolete.

If N is a positive integer the factorization problem consists on finding two integer numbers p and q numbers such that $N = p \cdot q$. The Shor algorithm transforms the problem from the search of p and q to the search of the period of a long sequence. This, thanks to modular exponentiation by repeated squaring and the quantum Fourier transform can theoretically be solved in polynomial time by a quantum computer.

The algorithm consist of six steps. However, just one of them requires a quantum algorithm, the fourth. The other ones can dealt with classical computations. The steps are [10]:

1. Check that N is odd, if it is even $\rightarrow$ 2 will be a factor and that it is not a power of a small integer like $3, 5, 7 \ldots$

2. Pick randomly an integer a between 1 and N.

3. Find the greatest common divisor between 1 and N. If it is greater than one congratulations, you have the factors and you are done. If not you to keep going.

4. Compute the smallest integer b, bigger than 1 that satisfies $a^p = 1(modN)$

5. Once you have b if it is odd or if $a^{\frac{p}{2}} = -1(modN)$ you have to go back to step 2 and to choose a different a. If not you have the solution.

6. The solution: the two factors are $\gcd(a^{\frac{p}{2}} + 1, N)$ and $\gcd(a^{\frac{p}{2}} - 1, N)$ where gcd denotes the greatest common divisor of two positive integers

6.1 How the quantum part works

The task of the quantum part of the algorithm is to find the period of the sequence that is modelled by the discrete function $f(x) = a^x mod(N)$. Once we have the period we can finish solving the problem with classical computations. The first step, as in any quantum algorithm, is to initialize the qubits. Then we have to

compute the action of the function $f(x)$ on the qubits. Finally, with the help of the quantum Fourier transform we find the sought for period needed to obtain the factors. The quantum circuit used to solve this problem is:

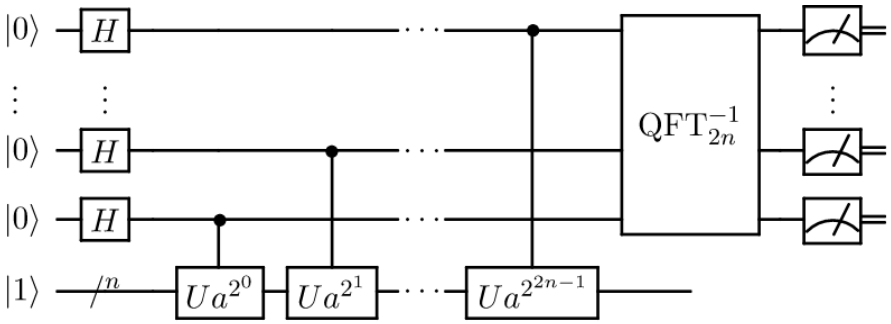


Figure 40: The Shor circuit [22]

Besides the well known Hadamard gates, we have to introduce some new gates to perform a number of new tasks. To implement the function in the circuit we have the quantum gates Ua^{2^0}, Ua^{2^1} etc. These are different for each case because the function depends on N and a and this changes every time. To compute the period we have to implement a quantum Fourier transform. In the figure this is labeled with QFT_{2n}^{-1}. It is composed by Hadamard and controlled phase shift gates.

The qubits are divided in two groups. The first one is called the L register and is initialized to $|0...0\rangle$. The second one, the M register is initialized to $|0...01\rangle$. The size of the registers depends on N. For L, the bigger the number of qubits, the more accuracy the measurements will have. To have the guarantee to find the period the size of the L register should satisfy $2^L \geq N^2$. However, the Shor algorithm can work with an smaller L and provide good enough results. The size of M should satisfy $2^M \geq N$ because M is used to express in binary the results of evaluating the function $f(x) = a^x mod(N)$ and they could be up to $N-1$ [10].

As shown in the figure the first gates acting on the qubits are the Hadamards on the L register. This leaves the L register in an equal superposition state

$$\psi_L = \frac{1}{\sqrt{2^L}} \sum_{x=0}^{2^{L-1}} |x\rangle \tag{140}$$

Then both qubit registers pass through the gates to compute $f(x)$. The L register controls the gates that act on the M register. If the L register were not in a superposition state, the act of controlling will not change the L register, it will change just the controlled register, the M qubits. However, the L register is on an equal superposition. Determining if the gate acts or not is like measuring: the superposition state collapses. This have as consequence that the L superposition state is modified while it controls the operation [25]. Now for every value of the L there is a value assigned to the function in the M. As the function is periodic there is just a finite number of some possibilities that can be obtained by measuring. However, as our focus is on the L register, the measurement of the M one is not

even necessary. This part is the main bottleneck of the algorithm: it is by far the slowest.

After the action of these gates the state of the L register becomes:

$$\psi_L = \frac{1}{\sqrt{2^L}} \sum_{x=0}^{2^{L-1}} |x\rangle |f(x)\rangle \tag{141}$$

Now we apply the quantum Fourier transform. this is the quantum analog of the classical Fourier discrete Fourier transform and is used in many quantum algorithms. This gate transforms a quantum state

$$\psi_x = \sum_{i=0}^{N-1} x_i |i\rangle \tag{142}$$

into another quantum state

$$\psi_y = \sum_{i=0}^{N-1} y_i |i\rangle \tag{143}$$

according to the formula

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j w^{jk} \tag{144}$$

where

$$w^{jk} = e^{2\pi i \frac{jk}{N}} \text{ [23].} \tag{145}$$

The circuit that implements the quantum Fourier transform is:

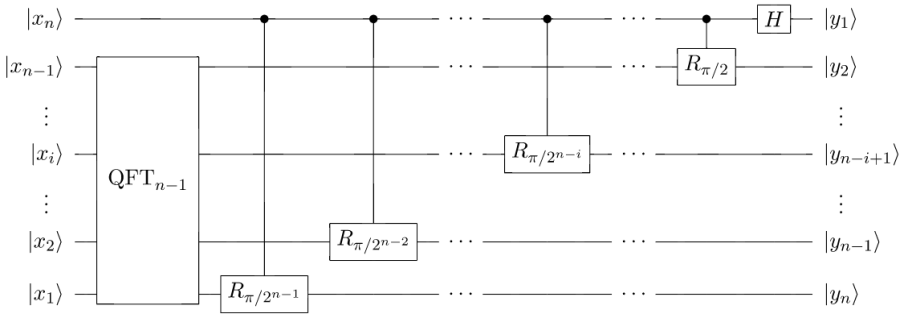


Figure 41: The Quantum Fourier transform circuit [26]

It is important to realize that the quantum Fourier transform flips the order of the qubits from its input to its output.

In our case the Fourier quantum transform converts the L register into [21]

$$\psi_L = \frac{1}{2^L} \sum_{x=0}^{2^{L-1}} \sum_{z=0}^{2^{L-1}} e^{2\pi i \frac{xy}{2^L}} |y\rangle |f(x)\rangle \,, \tag{146}$$

To simplify the computations it is convenient to reorder this expression as

$$\psi_L = \frac{1}{2^L} \sum_{x=0}^{2^{L-1}} \sum_{z=0}^{2^{L-1}} |y\rangle |f(x)\rangle \sum_{x:f(x)=z}^{2^{L-1}} e^{2\pi i \frac{xy}{2^L}}, \tag{147}$$

and transform the last sum to

$$\frac{1}{2^L} \sum_{x:f(x)=z}^{2^{L-1}} e^{2\pi i \frac{xy}{2^L}} = \frac{1}{2^L} \sum_b e^{2\pi i \frac{(x_0+rb)y}{2^L}} = \frac{e^{2\pi i \frac{x_0 y}{2^L}}}{2^L} \sum_b e^{2\pi i \frac{rby}{2^L}}, \tag{148}$$

Now we are ready to measure.[7]

Since f is a periodic function the probability is given by [24]

$$\left| \frac{1}{2^L} \sum_b e^{2\pi i \frac{rby}{2^L}} \right|^2 . \tag{149}$$

To extract the information from this result we have to make a continued fraction expansion of $\frac{y}{2^L}$ to approximate some $\frac{s}{p}$. If p satisfies the conditions of step 5 (not odd and $a^{\frac{p}{2}} \neq -1(modN)$) it will be the period of $f(x)$. Otherwise we have to try with another a.

Once we have found the period we just have to go throughstep 6 with a classical computer to obtain the answer that we are looking for.

6.2 Mathematica implementation of the Shor algorithm

For the Mathematica program we are going to focus also in the quantum part of the algorithm. Besides the general ingredients like the Hadamard gates the measure simulator and the initial state, we need gates to implement the function $f(x)$ and the controlled phase shift gates. For them we had to define new quantum gates and use some new Mathematica commands. I will first describe the gates needed to implement the function $f(x)$ and then the quantum Fourier transform using controlled phase shifts and Hadamard gates. Finally, once we have all of them we just plug them in the right order and act on the initial state. In the discussion we are going to first describe how the gates and the functions are implemented in Mathematica. Later we will discuss how we get the final state and the results of measuring on it.

The implementation that we are using for the quantum gates allows us to work with any number of qubits. This number is only limited by computer memory and the time needed to run the programs. In practice even the factorization of the number 15, that we discuss in the example presented below, takes a significant time. For this factorization we need three qubits in the L register and four in the M, making a total of seven. This means that all the transformation matrices associated with the quantum gates are going to be $2^7 \times 2^7 = 128 \times 128$. The circuit for our system looks like:

[7] We could go through this in more detail but is beyond the scope of this TFG and not very illuminating for our simulations.

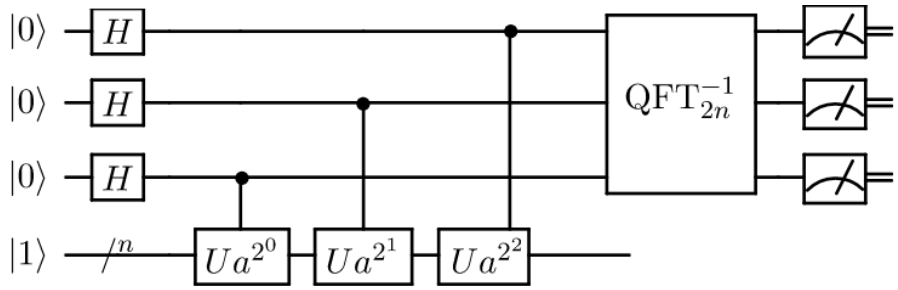


Figure 42: The Shor circuit [22]

To build our matrices we follow [10].

6.2.1 The function implementation

In the following we need to define two kinds of functions. The first type, that we denote as Cf, is a general routine that is used in the implementation of Ua^{2^0}, Ua^{2^1} and Ua^{2^2}. The function Cf depends on the following variables.

- l allows us to select if we want to compute, $a(\mod n)$, $a^2(\mod n)$ or $a^4(\mod n)$.
- m, n label matix entries. For the our seven qubit system they start at 1 and end at 128.
- a is the random number between 0 and N selected by the second step of the algorithm (classical part).
- numero is the number that we want to factor.
- numqubits is the number of qubits.

Additionally we need other internal variables. They are going to be defined locally inside a Module, a Mathematica command previously discussed for the measure program. In the present case we use:

- A This first expression computes the function f(x) for the three gates. It stores the values of $a(\mod n)$, $a^2(\mod n)$ and $a^4(\mod n)$
- L defines the size of the L qubit register, in this case 3.
- digitsn generates the digits of a binary number from n. We build this the same way as we built the first part of InitialStateGenerator.
- mmmm is the decimal expression of the binary number built with the digits of the previous step.
- fp Multiplies the number mmmm by $a(\mod n)$, $a^2(\mod n)$ or $a^4(\mod n)$ modulo number.
- fpbin Coverts the result of fp to binary.

- r is the decimal expression of the binary number built with the digits of the previous step.
- j tells us the position of the row where we have to put a 1.
- resultado Puts a one in the position j.

The whole function looks like:

```
Cf[l_, m_, n_, a_, numero_, numqubits_] := Module[{A, digitsn, fp, fpbin, j, L, mmmm, r, resultado},
                                             módulo
  A = {Mod[a, numero], Mod[a^2, numero], Mod[a^4, numero]};
       operación módulo  operación módulo  operación módulo
  L = 3;
  digitsn = Join[Table[0, {k, 1, numqubits - Length[IntegerDigits[n - 1, 2]]}], IntegerDigits[n - 1, 2]];
            junta  tabla                  longitud  dígitos de entero          dígitos de entero
  mmmm = Plus @@ Table[2^(p - 1) * digitsn[[numqubits - p + 1]], {p, 1, numqubits - L}];
         suma     tabla
  fp = Mod[A[[l + 1]] * mmmm, numero];
       operación módulo
  fpbin = Join[digitsn[[1 ;; 3]], Join[Table[0, {k, 1, numqubits - Length[IntegerDigits[fp, 2]] - L}],
          junta                   junta  tabla                   longitud  dígitos de entero
     IntegerDigits[fp, 2]]] ;
     dígitos de entero
  r = Plus @@ Table[2^(p - 1) * fpbin[[numqubits - p + 1]], {p, 1, numqubits}];
      suma     tabla
  j = Which[digitsn[[L - l]] == 0, n - 1, 2 > digitsn[[L - l]] > 0 && mmmm + 1 > numero, n - 1, 2 > digitsn[[L - l]]
      cuál
       > 0 && mmmm < numero, r];
  resultado = KroneckerDelta[j, m - 1]; resultado]
              delta de Kronecker
```

Figure 43: The Cf function

In terms of this function it is straightforward to build the matrices for Ua^{2^0}, Ua^{2^1} and Ua^{2^2}, that we denote f1, f2 and f3. The three of them are defined with the help of y Cf.

```
f1[a_, numero_, numqubits_] :=
 SparseArray[Table[Cf[0, m, n, a, numero, numqubits], {m, 1, 2^numqubits}, {n, 1, 2^numqubits}]]
 array disperso  tabla
f2[a_, numero_, numqubits_] :=
 SparseArray[Table[Cf[1, m, n, a, numero, numqubits], {m, 1, 2^numqubits}, {n, 1, 2^numqubits}]]
 array disperso  tabla
f3[a_, numero_, numqubits_] :=
 SparseArray[Table[Cf[2, m, n, a, numero, numqubits], {m, 1, 2^numqubits}, {n, 1, 2^numqubits}]]
 array disperso  tabla
```

Figure 44: The f functions

For our simulation we will consider two cases a=11, numero=15 and numqubits=7 and a=7, numero=15 and numqubits=7. In each case the gates Ua^{2^0}, Ua^{2^1} and Ua^{2^2} are given by 128×128 matrices whose entries are zeroes or ones. They are stored as spare matrices (Mathematica has special commands to handle them). As they are large it is not practical to show them in the usual form

however it is possible to generate a picture showing only the non-zero entries. In the following figures the black dots represent the position of the non-zero entries (ones). The upper-left location corresponds, as usual, to the (1, 1) entry in the matrices.

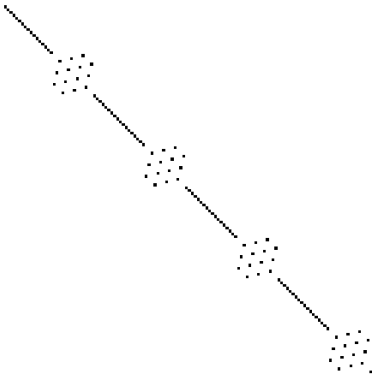


Figure 45: The $a \mod C$ gate matrix for a=11

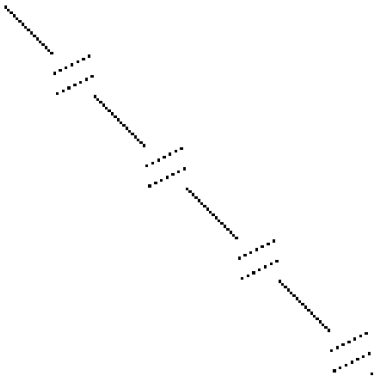


Figure 46: The $a \mod C$ gate matrix for a=7

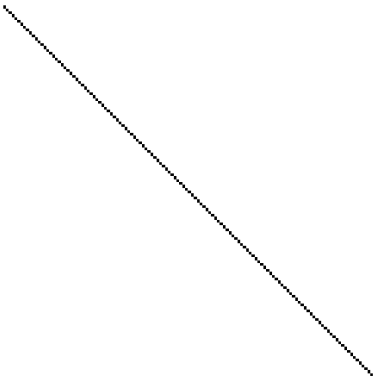


Figure 47: The $a \mod C$ gate matrix for a=11

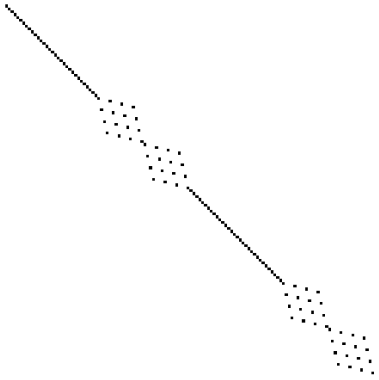


Figure 48: The $a^2 \mod C$ gate matrix for a=7

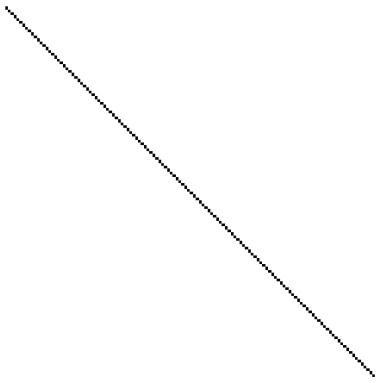


Figure 49: The $a^4 \mod C$ gate matrix for a=11 and a=7

6.2.2 The Fourier transform

For a three qubit system we can implement the quantum Fourier transform with a combination of controlled phase shift gates and Hadamards. They act on the L register as indicated in the following quantum circuit:

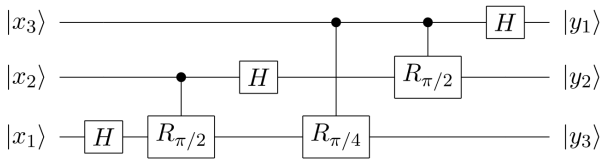


Figure 50: The QFT for three qubits [27]

We have already explained the implementation of the Hadamard gates so we have just to define a new function for the controlled phase shift.

The controlled phase shift gate takes as input the number of qubits, the phase shift and the position of the controlled and the controlling qubit. These matrices are going to be large diagonal matrices. The difference between them is where the non-trivial phases are placed, these determine which qubits undergo the phase shift and who is controlling them. As we did before we first determine the part involving the non-trivial phases and then build the whole matrix in terms of them. We

introduce two commands for this. The first one is called CRentrada and depends on the gate that controlling and the gate that is controlled. This information is imputed into the function as the coordinates of a vector called a. The first component refers to the controlling qubit an the second to the controlled. Then, as in the Cf, we have the labels m and n. They have the same function as in Cf. The other two variables of the function are θ (the phase shift angle expressed in radians) and the numqubits.

The function has the same structure as Cf. We use the command Module to build it. This time inside it we have:

- R is the matrix of the quantum gate corresponding to the controlled phase shift involving the qubit 1 and the qubit 2 in a two-qubit system

  It looks like:
  $$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\theta i} \end{pmatrix} \text{ [13].} \tag{150}$$

- digitsm converts the column location defined by m into a binary number.
- digitsn converts the row location defined by n into a binary number.
- resto extracts the information from the vector a to establish which qubit is controlling and which one is controlled.
- A combines the information from resto with digitsm to be used in resultado
- B combines the information from resto with digitsn to be used in resultado
- resultado writes the value of an entry as a product of Kronecker deltas and the 4×4 matrix given above. It uses the KroneckerDelta Mathematica command.

The whole function looks like:

```
CRentrada[a_, m_, n_, θ_, numqubits_] := Module[{R, digitsm, digitsn, resto, A, B, resultado},
                                          |módulo
  R = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, Exp[I θ]}};
                                                  |ex··· |número i
  digitsm = Join[Table[0, {k, 1, numqubits - Length[IntegerDigits[m - 1, 2]]}], IntegerDigits[m - 1, 2]];
            |junta |tabla                  |longitud |dígitos de entero          |dígitos de entero
  digitsn = Join[Table[0, {k, 1, numqubits - Length[IntegerDigits[n - 1, 2]]}], IntegerDigits[n - 1, 2]];
            |junta |tabla                  |longitud |dígitos de entero          |dígitos de entero
  resto = Flatten[ReplacePart[Table[k, {k, 1, numqubits}], {a[[1]] → {}, a[[2]] → {}}]];
          |aplana  |sustituye una··· |tabla
  B = Extract[digitsm, a[[1]]] * 2 + Extract[digitsm, a[[2]]] + 1;
      |extrae                       |extrae
  A = Extract[digitsn, a[[1]]] * 2 + Extract[digitsn, a[[2]]] + 1;
      |extrae                       |extrae
  resultado = Extract[R, {A, B}] * (Times @@ Table[KroneckerDelta[Extract[digitsm, resto[[k]]],
              |extrae               |multiplic··· |tabla |delta de Kronecker |extrae
        Extract[digitsn, resto[[k]]]], {k, 1, numqubits - 2}]); resultado];
        |extrae
```

Figure 51: The function CRentrada

The matrix implementing the controlled phase shift gates is

```
CR[a_, θ_, numqubits_] :=
 SparseArray[Table[CRentrada[a, j, k, θ, numqubits], {j, 1, 2^numqubits}, {k, 1, 2^numqubits}]]
 array disperso  tabla
```

Figure 52: The function CR

With these ingredients we can implement the quantum Fourier transform subroutine for a three qubit system. The Mathematica code implementing this is simply

```
QFT = (Hadamard[7, 3].CR[{2, 3}, π / 2, 7]
    .Hadamard[7, 2].CR[{3, 1}, π / 4, 7]
    .CR[{2, 1}, (π / 2), 7].Hadamard[7, 1])
```

Figure 53: The 3 qubits quantum Fourier transform in Mathematica

We have to use the matrices for seven qubits because our total number of qubits is seven but these gates only act on the first three, the ones for the L register. The other ones remain in the same state. That will be shown in the simulations.

6.3 Factoring 15 with our program

As said before our goal is to factor 15. We are going to follow all the steps of the algorithm, even the classical part of the computation to emulate how this computation would be carried out. The first step is to check if the number that we want to factor is even or a power of a small integer. We can easily see that we 15 is neither even nor a power of a small integer. The next step is to pick a number a between 1 and 15. Here if the greatest common divisor between a and 15 is bigger than one, the problem is solved. Therefore to avoid "solving" the problem right away, we cannot choose 3, 5, 9, 10 or 12 because the greatest common divisor between these numbers and 15 is bigger than 1. Otherwise we could not test the algorithm with our simulation. This leaves us with 2, 4, 7, 8, 11, 13 and 14. Once we have a we substitute it as well as 15 in the quantum gates related with $f(x)$. With this we are ready to perform the simulation.

To see all the steps inside the quantum part, we are going to compute it in two phases. First we are going to use the Ua^{2^0}, Ua^{2^1} and Ua^{2^0} gates (function action) and then we are going to "pass the qubits" through the quantum Fourier transform. After the function action, if we measure we have to obtain two things: the values of $f(x)$ in the qubits corresponding to the M register and random numbers in the L register. When we compute the quantum Fourier transform this changes. In the M register we still have the same outputs. However, now from the L register we can extract the frequencies of the $f(x)$. From them we compute the period. Once we arrive this point, the quantum part is over and we just have to execute steps 5 and 6 to obtain the factors of 15. We are going to go through the Shor simulation

for two different values of a. First $a = 11$ and then $a = 7$. The other numbers are going to have similar behaviour, depending on the period $f(x)$ (2 in the case of 11 and 4, for 7).

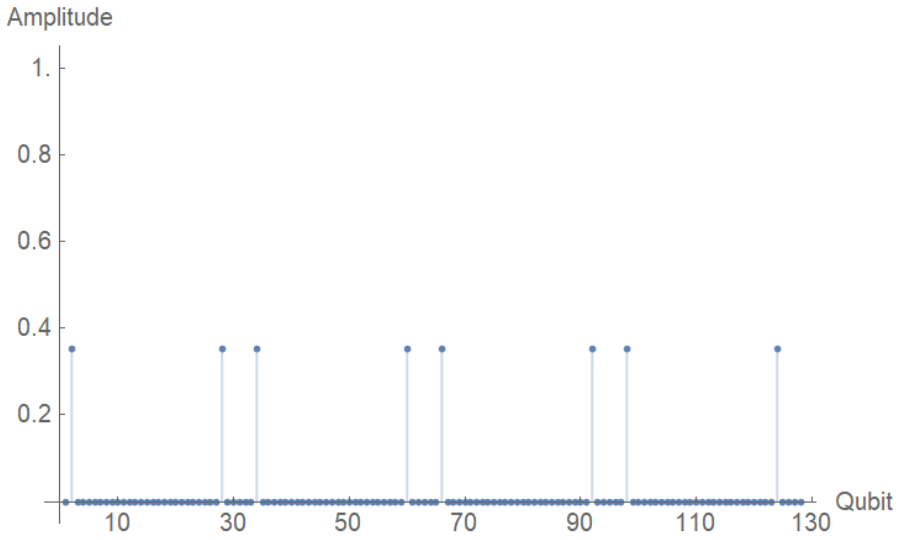
6.3.1 Factoring 15 with a=11

We are going to start with $a = 11$. As discussed before, this number satisfies the conditions specified in the steps 2 and 3. To compute the first quantum part we need the following Mathematica line of code:

```
ψ = matriz3.matriz2.matriz1.Hadamard[7, 3].Hadamard[7, 2].Hadamard[7, 1].InitialStateGenerator[2, 7]
```

Figure 54: Implementation of $f(x)$ for the Shor algorithm with a=11 and N=15

The elements matriz1, matriz2 and matriz3 correspond to the sparse matrices of f1, f2 and f3 when we substitute in them $a = 11$ and $N = 15$ and numqubits= 7. However these are not the only variables that we need. We have to include EstadoInicial and the Hadmards too. The difference between a, N, and the rest of them is that in the last group they remain constant for different vaules of a and N until we change the number of qubits of the system (the size of the registers L and M). When Mathematica computes the expression we obtain the following state:



To see the effects described in [10] we have to measure a significant amount of times. A batch of one hundred measurements performed on this state gave the following results:

```
FinalCounter[100, 7]
⟨| 1011011 → 16, 1111011 → 14, 1000001 → 14, 0011011 → 10, 0000001 → 13, 1100001 → 15, 0100001 → 14, 0111011 → 4 |⟩
```

Here we can see how the last four digits of the state, the M register, are either 0001 (one in decimal) or 1011 (eleven in decimal). These are the values of the

function $f(x) = 11^x(\mod 15)$ for discrete imputs x (for $x = 1$, $x = 2, \ldots$). We also see that the L register varies randomly between 000, 001, ... and 111 as it should.

Then we apply the quantum Fourier transform. This part remains unchanged for all the possible numbers that use this registers size. By multiplying by the unitary matirx that implements the quantum Fourire transform for a three-qubit system we finally get

```
ψ = Hadamard[7, 3].CR[{2, 3}, π / 2, 7].Hadamard[7, 2].CR[{3, 1}, π / 4, 7].
  CR[{2, 1}, π / 2, 7].Hadamard[7, 1].matriz3.matriz2.matriz1.Hadamard[7, 3].
  Hadamard[7, 2].Hadamard[7, 1].InitialStateGenerator[2, 7]
```

This result is:

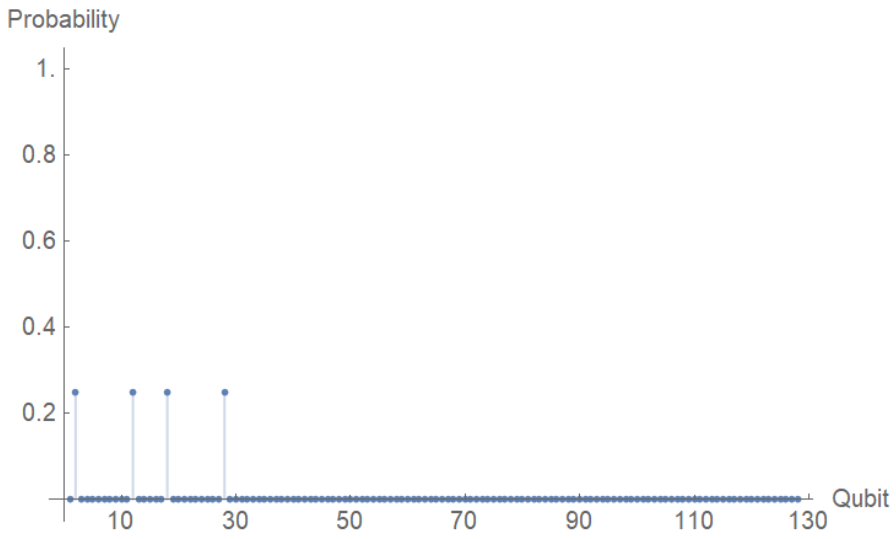


Figure 55: Final state for the Shor algorithm, N=15 and a=11

By measuring one hundred times on this state, we obtained:

```
FinalCounter[100, 7]

⟨| 0010001 → 22, 0000001 → 21, 0001011 → 26, 0011011 → 31 |⟩
```

Figure 56: Measurments for the Shor algorithm, N=15 and a=7

For the appropriate interpretation of the measurements we have to consider several things. The first one is that we have to read the L register from right to left. Also that the solution is little bit hidden: We have to compute $\frac{\widehat{x}}{2^L}$, being $\widehat{x}$ the numerical results of the L register transformed from binary to decimal. Then we have to find some p, equal for all the options, such that $\frac{\widehat{x}}{2^L} = \frac{s}{p}$ where s any integer and p the result we are looking for. We compile all the information in the following table

Final State	N° of measurements	L register	L register decimal	$\frac{\hat{x}}{2^L}$	$\frac{s}{p}$
0010001	22	100	4	$\frac{4}{8}$	$\frac{1}{2}$
0000001	21	000	0	$\frac{0}{8}$	$\frac{0}{2}$
0001011	26	000	0	$\frac{0}{8}$	$\frac{0}{2}$
0011011	31	100	4	$\frac{4}{8}$	$\frac{1}{2}$

Table 1: Results for N=15 and a=11

From here we can extract the conclusion that the period is 2. Now we have to check if p is odd or if it satisfies $a^{\frac{p}{2}} = -1(\mod N)$. If any one of these condition hold, then we have to pick another a. In our case p is not odd and $11^1 \neq -1(\mod 15)$. Therefore we can go to step 6. The greatest common divisor between 15 and $a^{\frac{p}{2}} + 1 = 11 + 1 = 12$ is 3 and between 15 and $a^{\frac{p}{2}} - 1 = 11 - 1 = 10$ is 5. We have found that $15 = 5 \times 3$.

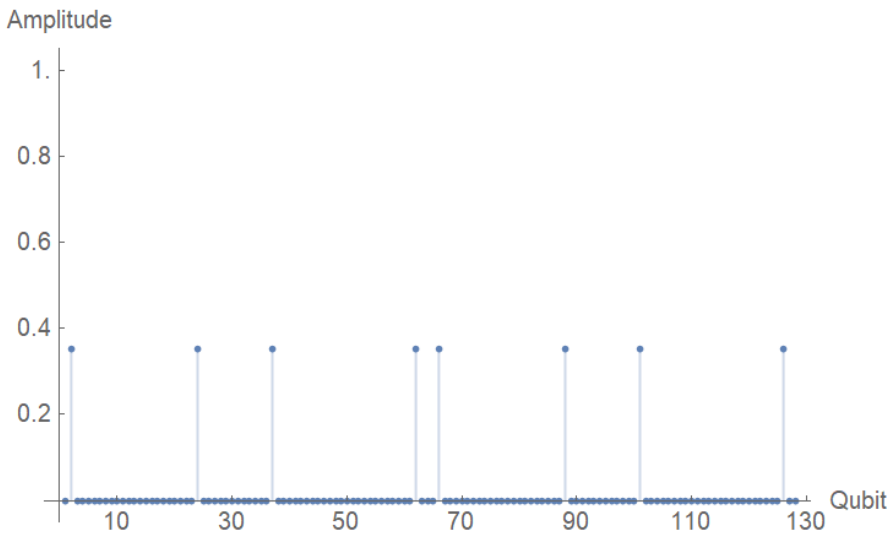
6.3.2 Factoring 15 with a=7

Now we consider to the case $a = 7$. As well as $a = 11$, this number satisfies the conditions of steps 2 and 3 so we can move forward. We proceed as before

```
ψ = matriz3.matriz2.matriz1.Hadamard[7, 3].
  Hadamard[7, 2].Hadamard[7, 1].InitialStateGenerator[2, 7]
```

Figure 57: Implementation of $f(x)$ in the Shor algorithm with a=7 and N=15

As before, $a = 7$ and $N = 15$ have to be introduced in f1, f2 and f3. The other variables remain as in the previous example. When we execute it we obtain the following state:



with matriz1, matriz2 and matriz3 corresponding to the sparse matrices of f1, f2 and f3 when we plug in them $a = 7$, $N = 15$ and numqubits= 7.

As we did before now we measure one hundred times on this state

```
FinalCounter[100, 7]
```

```
⟨| 0010111 → 9, 1100100 → 20, 1000001 → 10, 0000001 → 16, 1010111 → 8, 1111101 → 10, 0111101 → 8, 0100100 → 19 |⟩
```

Here we can see how the last four digits of the state, the M register, are either 0001 (one in decimal), 0100 (four in decimal), 0111 (seven in decimal) or 1101 (thirteen in decimal). These are the values of the function $f(x) = 7^x(\mod 15)$ for $x = 1$, $x = 2, \ldots$. As before we see that the L register varies randomly between 000, 001, ... and 111.

We apply now the quantum Fourier transform. We do not have to modify it because it does not depend on a.

```
ψ = Hadamard[7, 3].CR[{2, 3}, π / 2, 7].Hadamard[7, 2].CR[{3, 1}, π / 4, 7].
  CR[{2, 1}, π / 2, 7].Hadamard[7, 1].matriz3.matriz2.matriz1.Hadamard[7, 3].
  Hadamard[7, 2].Hadamard[7, 1].InitialStateGenerator[2, 7]
```

This result is:

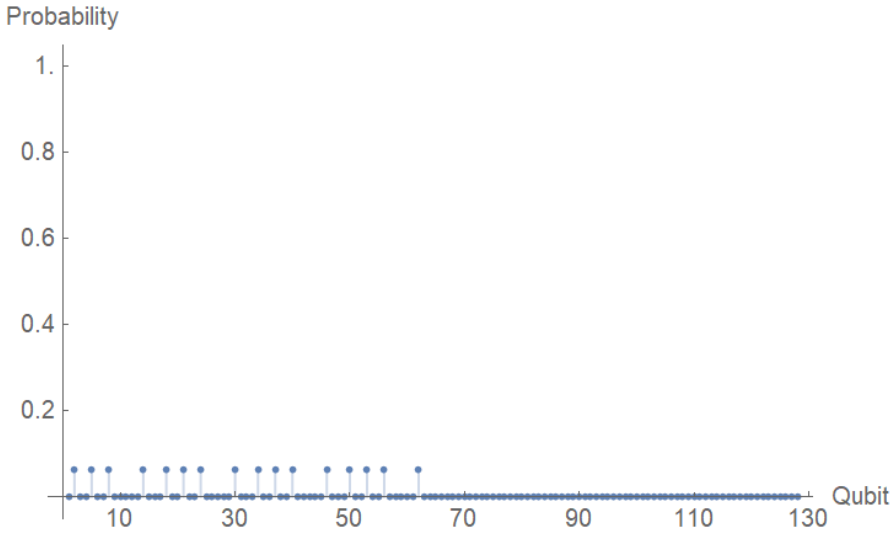


Figure 58: Final state for the Shor algorithm, N=15 and a=7

When we measure 100 times this state, we obtain:

```
⟨| 0011101 → 10, 0000111 → 10, 0000100 → 7, 0101101 → 9, 0100100 → 7,
 0010111 → 5, 0001101 → 6, 0100111 → 5, 0110001 → 5, 0010100 → 9, 0100001 → 5,
 0010001 → 3, 0110100 → 6, 0111101 → 5, 0000001 → 4, 0110111 → 4 |⟩
```

Figure 59: Measurements for the Shor algorithm, N=15 and a=11

We have to consider the same things as we did in the previous example with $a = 11$. In this case the resulting table is:

Final State	N° of measurements	L register	L register decimal	$\frac{\hat{x}}{2^L}$	$\frac{s}{p}$
0011101	10	100	4	$\frac{4}{8}$	$\frac{2}{4}$
0000111	10	000	0	$\frac{0}{8}$	$\frac{0}{4}$
0000100	7	000	0	$\frac{0}{8}$	$\frac{0}{4}$
0101101	9	010	2	$\frac{2}{8}$	$\frac{1}{4}$
0100100	7	010	2	$\frac{2}{8}$	$\frac{1}{4}$
0010111	5	100	4	$\frac{4}{8}$	$\frac{2}{4}$
0001101	6	000	0	$\frac{0}{8}$	$\frac{0}{4}$
0100111	5	010	2	$\frac{2}{8}$	$\frac{1}{4}$
0110001	5	110	6	$\frac{6}{8}$	$\frac{3}{4}$
0010100	9	100	4	$\frac{4}{8}$	$\frac{2}{4}$
0100001	5	010	2	$\frac{2}{8}$	$\frac{1}{4}$
0010001	3	100	4	$\frac{4}{8}$	$\frac{2}{4}$
0110100	6	110	6	$\frac{6}{8}$	$\frac{3}{4}$
0111101	5	110	6	$\frac{6}{8}$	$\frac{3}{4}$
0000001	4	000	4	$\frac{0}{8}$	$\frac{0}{4}$
0110111	4	110	6	$\frac{6}{8}$	$\frac{3}{4}$

Table 2: Results for N=15 and a=7

From here we can extract the conclusion that the period is 4. Now we have to check if p is odd or if it satisfies $a^{\frac{p}{2}} = -1(\mod N)$. If one of those condition hold, then we need to pick another a. In our it is clear the p is not odd and that $7^2 = -1(\mod 15)$ is not true. Therefore we can go to step 6. The greatest common divisor between 15 and $a^{\frac{p}{2}} + 1 = 7^2 + 1 = 50$ is 5 and between 15 and $a^{\frac{p}{2}} - 1 = 7^2 - 1 = 48$ is 3, so we conclude again that $15 = 5 \times 3$.

6.4 Conclusions and comments on the Shor algorithm

As commented before the practical implementation of the Shor algorithm in a quantum computer would change in a significant way the way secure information is handled and transmitted. The way cryptography works would have to change. In any case, until now researchers have been able just to factorize two numbers: 15 and 21 [10].

From my perpective this is a much more complicated algorithm than the Grover one. It was very challenging to make the simulation work. The hardest part was the implementation of the function $f(x)$, not just its implementation in Mathematica, in particular the implementation of controlled gates in the general case. In the world that we are used to if you control something with another thing, the normal procedure is that the controlling part is not alterated while the controlled is.

7 Final Conclusions and Comments

Doing this TFG I have expand my knowledge on a new subject: Quantum Physics. It is a very interesting field but a very hard one. Everything goes beyond our imagination and there are plenty of things that do not even have a classical counterpart. This makes very difficult to understand many important concepts. From this huge field this TFG has focused on quantum computation, more precisely on the algorithms that one day may run in quantum computers. I expect that there will be very important developments on this subject in the near future and I hope that the work that I have done here will help me understand them.

Once I understood the concepts behind the quantum computers, in particular the concept of qubit, everything become more familiar and therefore easier. I was surprised by the physical phenomenon that explains the power of quantum computation: the superposition of states and entanglement. These are powerful tools that allow the quantum computers to solve problems in a completely different way. They could make it possible to solve problems that we cannot solve with the current means or we can only approach in an approximate way. Some sample problems are the description of molecules that I mentioned before [4] or optimisation problems.

We must keep in mind, in any case, that before we are able to solve these problems we have to build large quantum computers involving many qubits. Right now many companies are trying its best to overcome the technical difficulties of this kind of technology. For example IBM, as mentioned before, is working in quantum computers with supercooled superconductors as qubits. They have been able to create a stable 5 qubits computer. This computer is accessible through the Internet and where you can run your own algorithms with the real computer [28]

Despite what you may think, when real big quantum computers arise, classical computer might still have an important role to play. In fact, I think that the best option will be to combine quantum and classical computation as it is done in the Shor algorithm. This allows us to take advantage of the strengths of each type of approach.

Finally I would recommend you to keep an eye on the quantum computation and computers because, sooner than later this field is going to amaze us, and change for ever the way we imagine the future.

References

[1] L. E. Ballentine *Quantum Mechanics: A Modern Development* World Scientific Publishing Company; 2 edition (November 5, 2014)

[2] R. Eisberg and R. Resnick *Quantum Physics of Atoms, Molecules, Solids, Nuclei, and Particles* John Wiley & Sons; 2nd Edition edition (10 April 1985)

[3] Commons.wikimedia.org. (2017). File:Stern-Gerlach experiment svg.svg - Wikimedia Commons. [online] Available at: https://commons.wikimedia.org/wiki/File:Stern-Gerlach_experiment_svg.svg [Accessed 8 Mar. 2017].

[4] J. Chow *The future of supercomputers? A quantum chip colder than outer space* TED Institute (10 December 2015)

[5] Visual Science, Nature, vol 543 (23 March 2017).

[6] Commons.wikimedia.org. (2017). File:Bloch Sphere.svg - Wikimedia Commons. [online] Available at: https://commons.wikimedia.org/wiki/File:Bloch_Sphere.svg [Accessed 14 May 2017].

[7] Commons.wikimedia.org. (2017). File:OR from NAND.svg - Wikimedia Commons. [online] Available at: https://commons.wikimedia.org/wiki/File:OR_from_NAND.svg [Accessed 10 Apr. 2017].

[8] Commons.wikimedia.org. (2017). File:Deutsch-Jozsa Algorithm.svg - Wikimedia Commons. [online] Available at: https://commons.wikimedia.org/wiki/File:Deutsch-Jozsa_Algorithm.svg [Accessed 12 Apr. 2017].

[9] Commons.wikimedia.org. (2017). File:CNOT gate.svg - Wikimedia Commons. [online] Available at: https://commons.wikimedia.org/wiki/File:CNOT_gate.svg [Accessed 19 Jun. 2017].

[10] D. Candela *Undergraduate computational physics projects on quantum computing* American Journal of Physics 83, 688 (2015)

[11] Commons.wikimedia.org. (2017). File:Grovers algorithm.svg - Wikimedia Commons. [online] Available at: https://commons.wikimedia.org/wiki/File:Grovers_algorithm.svg [Accessed 14 May 2017].

[12] E. Strubell *An Introduction to Quantum Algorithms*COS498 Chawathe Spring 2011

[13] A. Galindo and M. A. Martín-Delgado *Information and computation: Classical and quantum aspects* (8 May 2002)

[14] N. D. Mermin *From Cbits to Qbits: Teaching computer scientists quantum mechanics* (22 July 2002)

[15] L. K. Grover *A fast quantum mechanical algorithm for database search* (May 1996)

[16] J. Wright and T. Tseng *Lecture 4: Grover's Algorithm* (21 September 2015)

[17] En.wikipedia.org. (2017). Grover's algorithm. [online] Available at: https://en.wikipedia.org/wiki/Grover%27s_algorithm [Accessed 10 Sep. 2017].

[18] G. Brassard, P. HØyer and A. Tapp (1998). *Quantum Algorithm for the Collision Problem* (1 May 1997)

[19] WatersTechnology.com. (2017). Australian Public and Private Sectors Join Forces for Quantum Computing - WatersTechnology.com. [online] Available at: http://www.waterstechnology.com/industry-issues-initiatives/3413516/australian-public-and-private-sectors-join-forces-for-quantum-computing [Accessed 5 Sep. 2017].

[20] E. W. Weisstein *Number Field Sieve* From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/NumberFieldSieve.html

[21] P. W. Shor *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer* AT&T Research (25 Jan 1996)

[22] Commons.wikimedia.org. (2017). File:Shor's algorithm.svg - Wikimedia Commons. [online] Available at: https://commons.wikimedia.org/wiki/File:Shor%27s_algorithm.svg [Accessed 17 Aug. 2017].

[23] D. Bacon *The Quantum Fourier Transform and Jordan's Algorithm* University of Washington (Winter 2006)

[24] L. LIN and M. Loretz *Quantum Algorithms in NMR Experiments* ETH Zurich (25 May 2012)

[25] Research, I. (2017). IBM Q - Quantum Experience. [online] Quantumexperience.ng.bluemix.net. Available at: https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=8443c4f713521c10b1a56a533958286b&pageIndex=1 [Accessed 12 Sep. 2017].

[26] Commons.wikimedia.org. (2017). File:Quantum Fourier transform on n qubits.svg - Wikimedia Commons. [online] Available at: https://commons.wikimedia.org/wiki/File:Quantum_Fourier_transform_on_n_qubits.svg [Accessed 15 Aug. 2017].

[27] Commons.wikimedia.org. (2017). File:Quantum Fourier transform on three qubits.svg -
Wikimedia Commons. [online] Available at:
https://commons.wikimedia.org/wiki/File:Quantum_Fourier_transform_on_three_qubits.svg [Accessed 17 Aug. 2017].

[28] Research, I. (2017). IBM Q - Quantum Experience. [online]
Quantumexperience.ng.bluemix.net. Available at:
https://quantumexperience.ng.bluemix.net/qx/editor [Accessed 20 Sep. 2017].